Improved reconstruction attacks using range query leakage

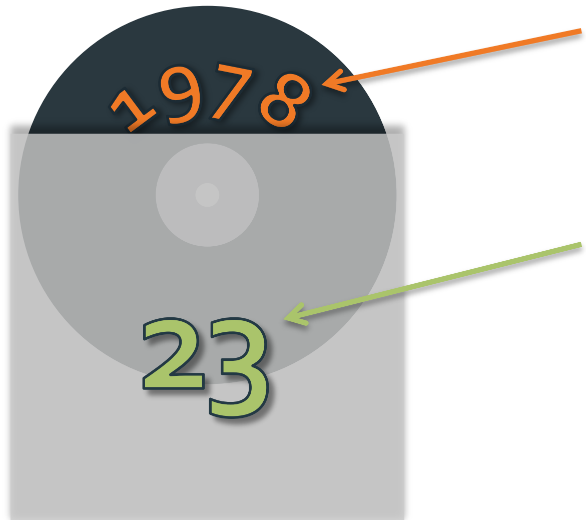
Marie-Sarah Lacharité
Brice Minaud
Kenny Paterson

Information Security Group

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Application Setting

Storing Records in the Cloud

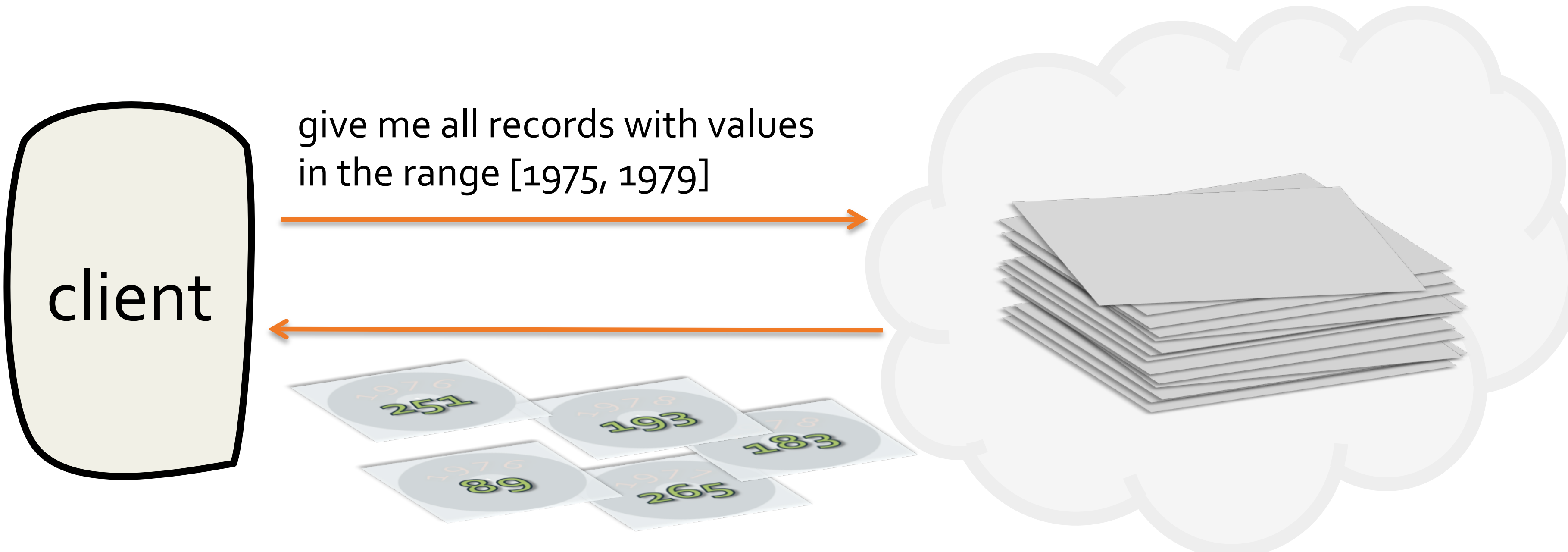


value of record
(*N* possible values)

record identifier
(unique)

R records

Application Scenario



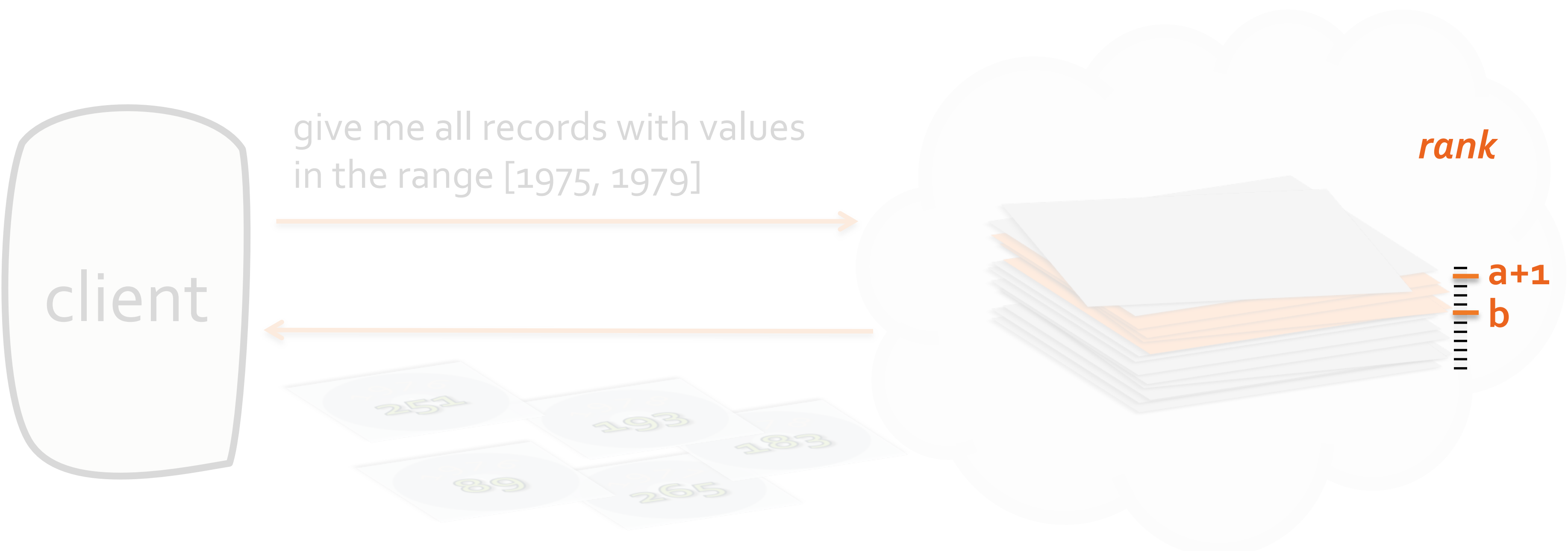
Access Pattern Leakage

record identifiers

{251, 89, 193, 265, 183}

OPE, ORE schemes, POPE, [HK16], Blind seer, [Lu12], [FJKNRS15],…

Access Pattern Leakage and Rank Leakage



record identifiers

{251, 89, 193, 265, 183}

FH-OPE, Lewi-Wu, Arx, Cipherbase, EncKV,…

Assumptions

1. Data is **dense:** all values appear in at least one record.
2. Queries are **uniformly distributed**.

Target: full reconstruction: find the value associated with each record.

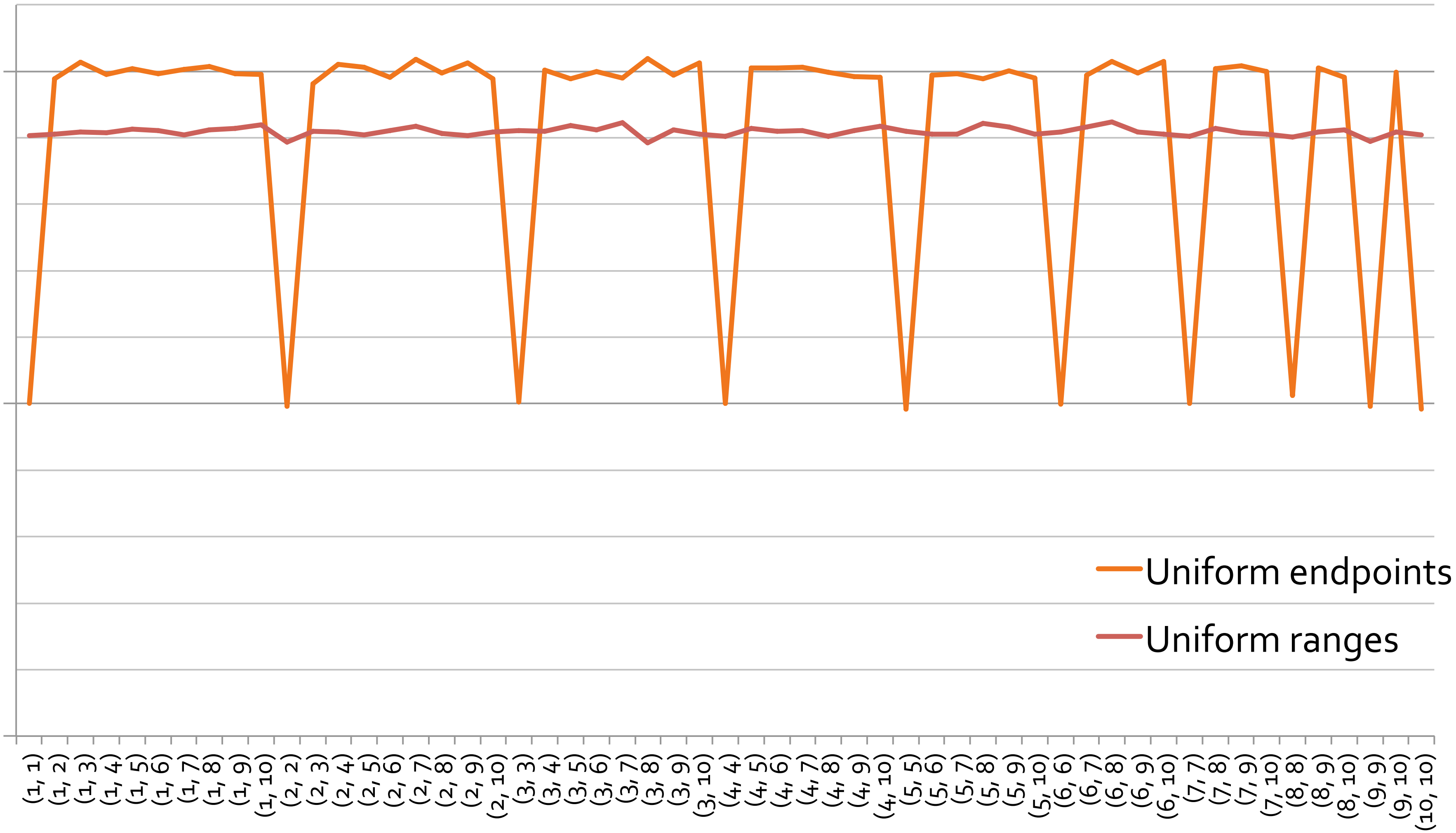
Best previous result (Kellaris et al., CCS 2016):

Full reconstruction by analysing access pattern leakage from $O(N^2 \log N)$ queries.

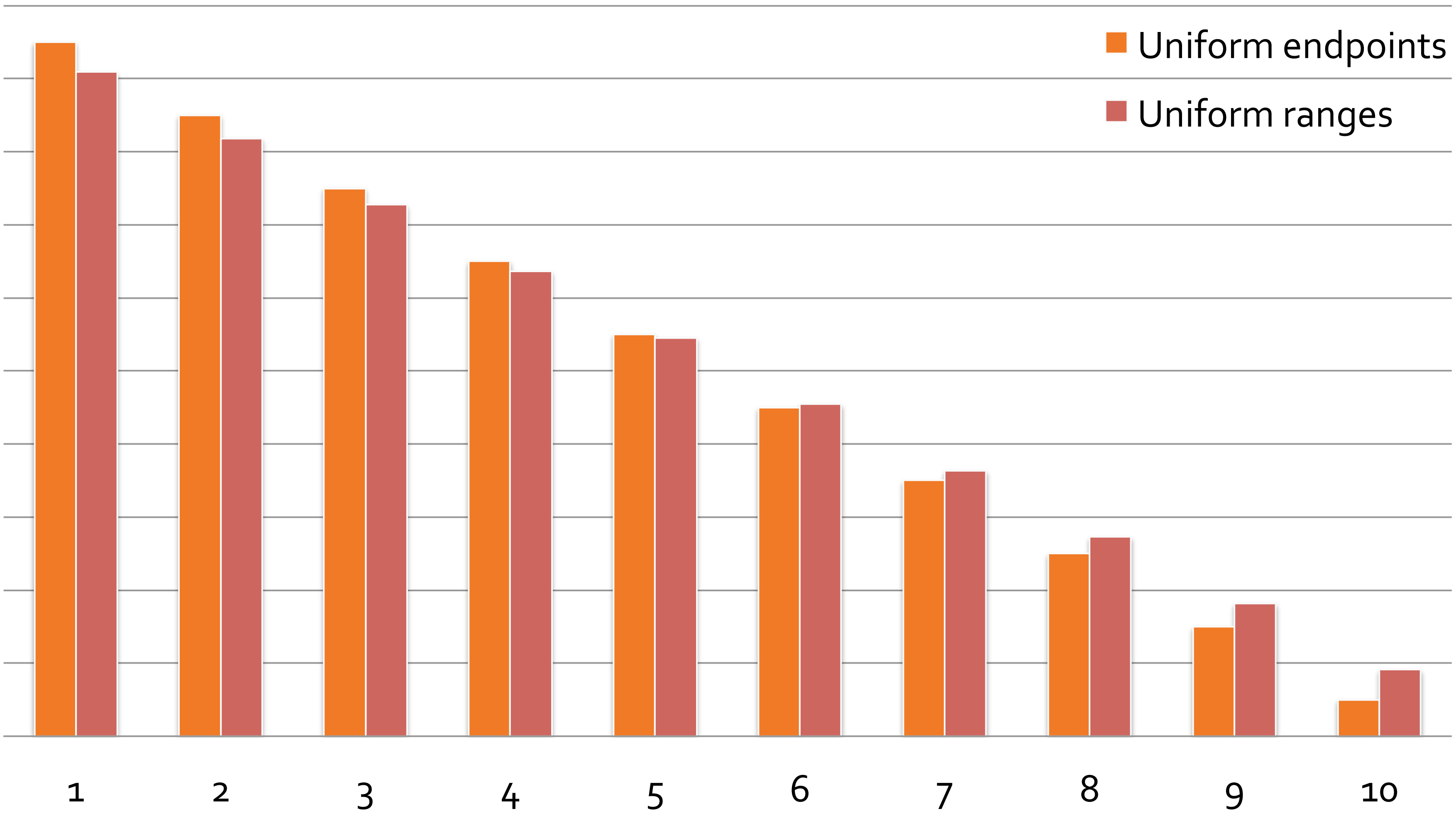
Our Main Results (eprint 2017/701)

- Full reconstruction with $O(N \log N)$ queries
  - in fact, expected $N \cdot (3 + \log N)$.
- Approximate reconstruction with relative accuracy ε from $O(N \cdot (\log 1/\varepsilon))$ queries
  - in fact, expected $5/4 \cdot N \cdot (\log 1/\varepsilon) + O(N)$.
- Approximate reconstruction using an *auxiliary distribution* and rank leakage.
  - more efficient in practice, evaluation via simulation.
  - applies in the non-dense case too, giving a new attack on OPE/ORE schemes.

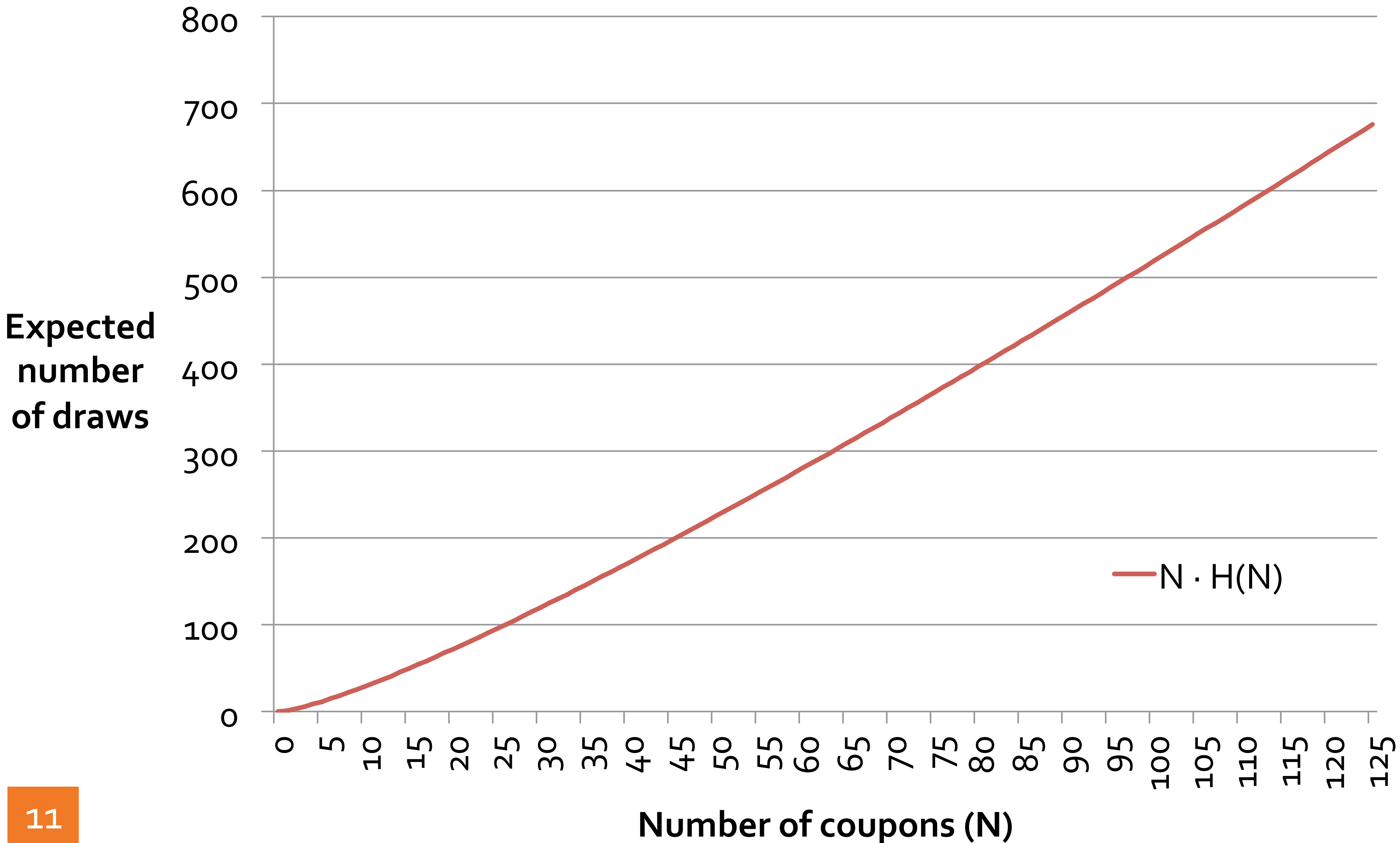
Uniform Queries: Uniform Endpoints vs. Uniform Ranges (N=10)



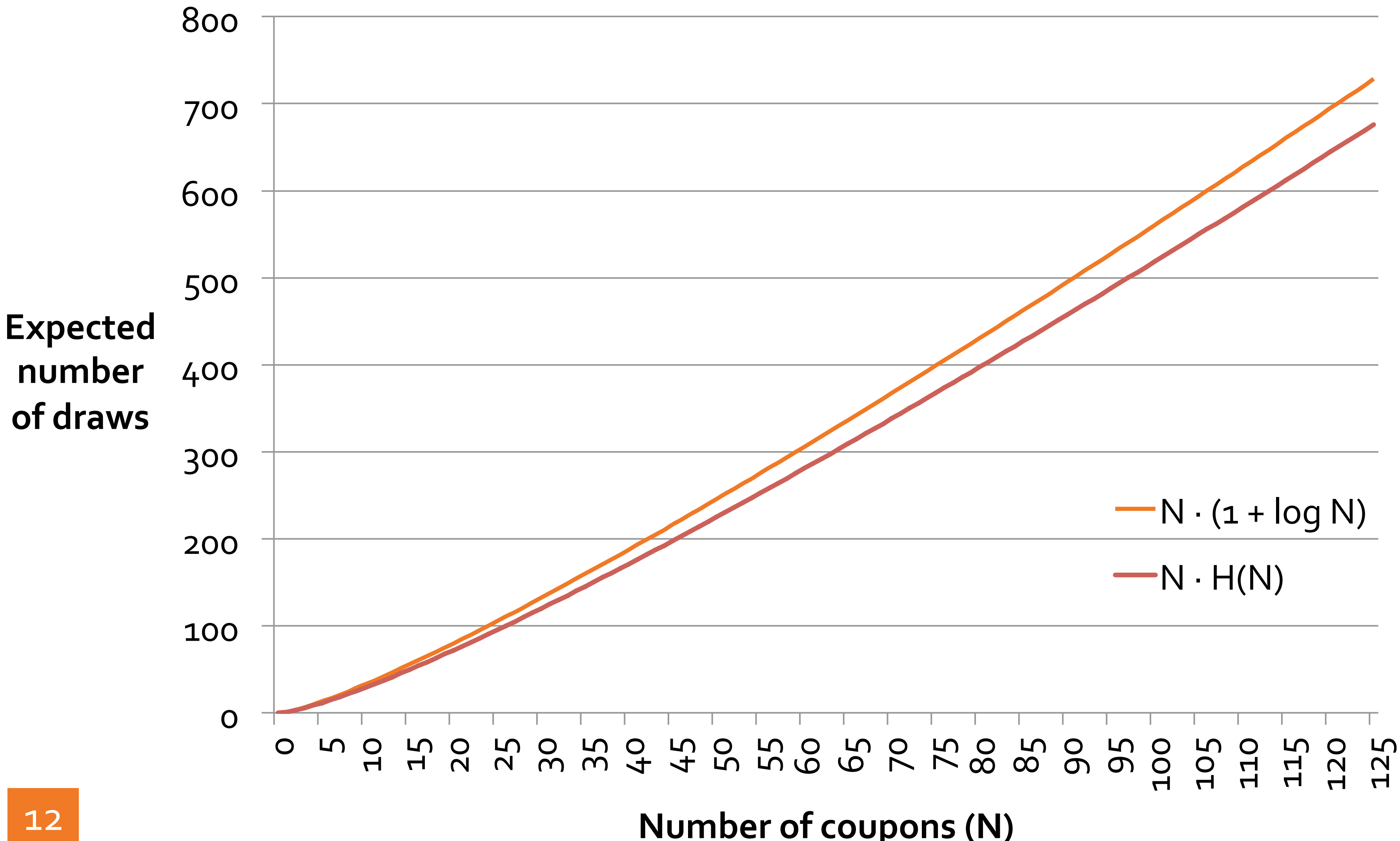
Distribution of Left Endpoints: Uniform Endpoints vs. Uniform Ranges (N=10)



Coupon Collector's Problem



Coupon Collector's Problem



Attack 1: Full Reconstruction

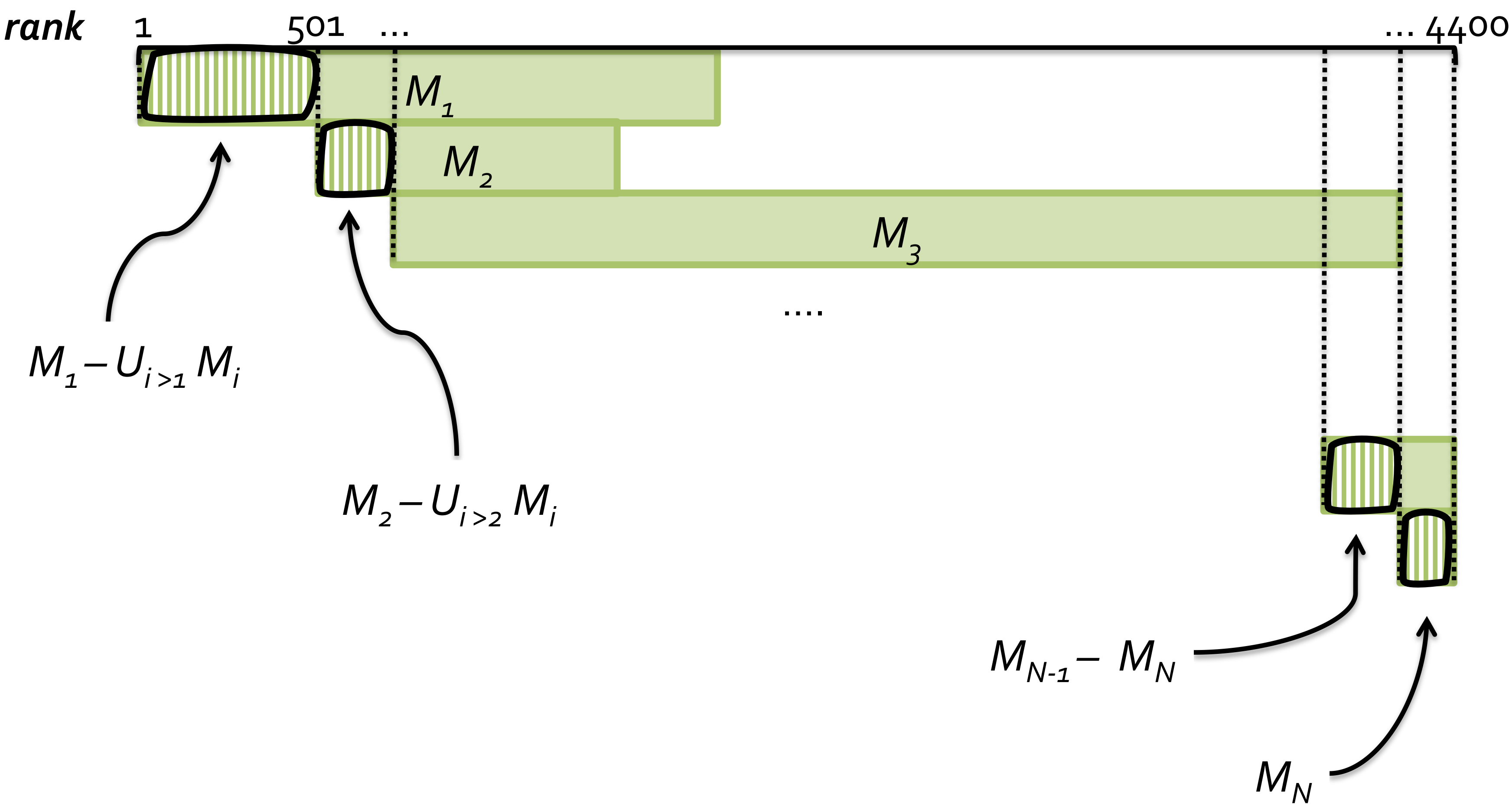
Motivating Example (with Rank Leakage)

- Suppose **left endpoints** of query intervals are chosen **uniformly at random.**
- Wish to observe at least 1 query with each of the N possible left endpoints.
- Expected number of queries needed is at most $N \cdot (1 + \log N)$.

hidden	*leaked*		
[x,y]	**a = rank(x-1)**	**b = rank(y)**	**matching IDs**
[20,25]	1300	1500	M_{20}
[1,18]	0	1200	M_{1}
[55,125]	3100	4400	M_{55}
[2,10]	500	800	M_{2}
[7,98]	700	4200	M_{7}

relabelled for convenience

Motivating Example (with Rank Leakage)



Full Reconstruction (with Rank Leakage)

- Now suppose queries have **ranges** chosen **uniformly at random.**
- We present a data-optimal algorithm (fails ⇨ full reconstruction is impossible).
- Expected number of sufficient queries is at most

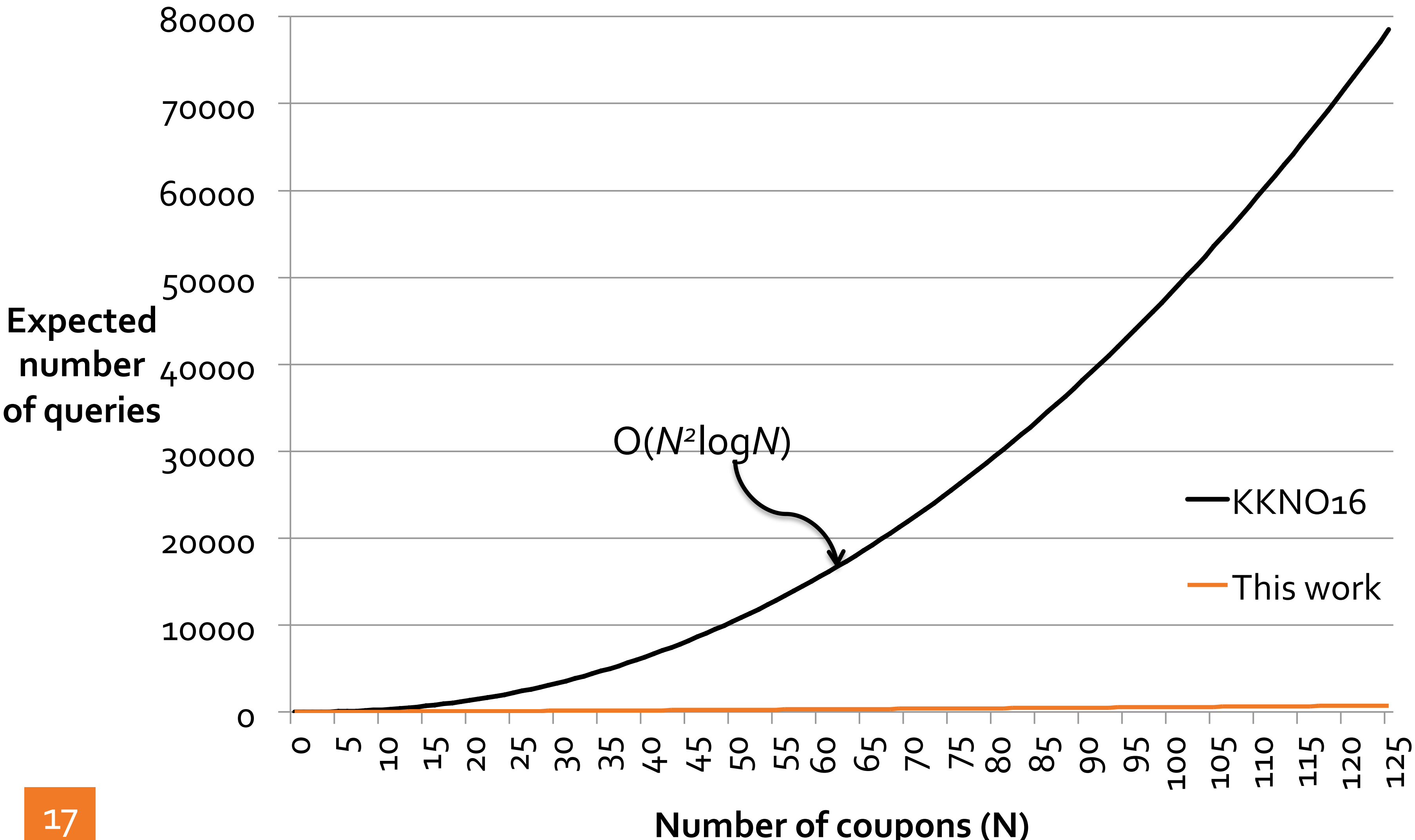
$$N \cdot (2 + \log N) \text{ for } N \geq 27.$$

- Main idea: partition, then sort (easy with rank leakage, harder without).
- Expected number of necessary queries is at least

$$1/2 \cdot N \cdot \log N - O(N)$$

for any algorithm.

Full Reconstruction (with Rank Leakage)



Full Reconstruction (with Rank Leakage): Partitioning Step

record ID	matched query?						
	1	2	3	4	5	6	7
20	✓	✓	✗	✗	✓	✗	✗
⇨ 23	✓	✓	✗	✗	✓	✓	✓
29	✗	✓	✓	✗	✗	✓	✗
89	✗	✓	✓	✗	✓	✓	✗
⇨ 193	✓	✓	✗	✗	✓	✓	✓
…							

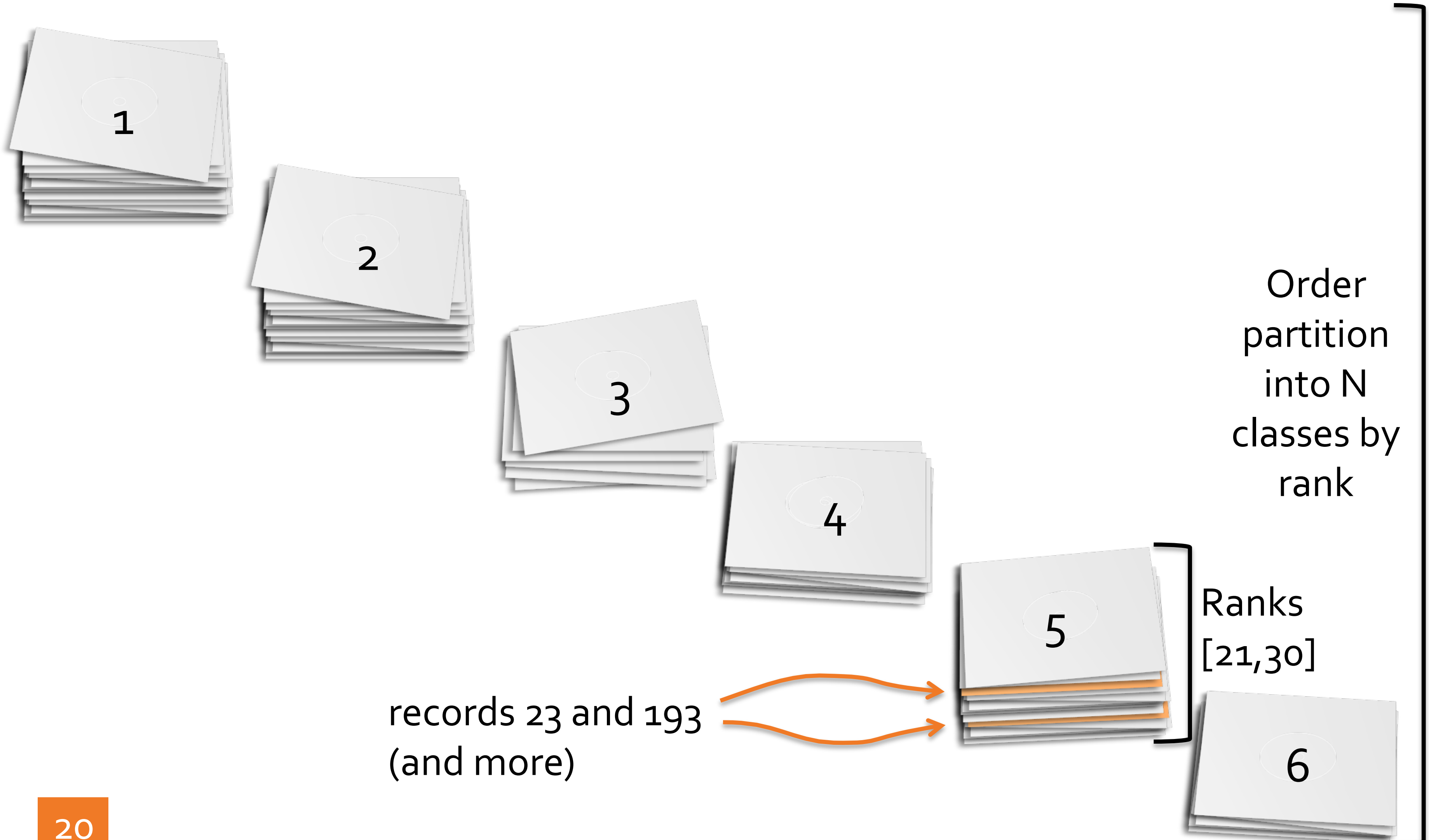
- Equality of matching defines a **partition** of records.
- Records in same class of partition cannot be distinguished.
- For complete reconstruction, we need *N* classes – one class per value.

Full Reconstruction (with Rank Leakage): Partitioning Step

record ID	matched query?						
	1	2	3	4	5	6	7
20	✓	✓	✗	✗	✓	✗	✗
23	✓ [1,100]	✓ [18,82]	✗	✗	✓ [16,96]	✓ [16,30]	✓ [21,61]
29	✗	✓	✓	✗	✗	✓	✗
89	✗	✓	✓	✗	✓	✓	✗
193	✓	✓	✗	✗	✓	✓	✓
…							

Can also deduce from rank leakage that, e.g., records 23 and 193 have ranks in [21,30], by intersecting rank intervals.

Full Reconstruction (with Rank Leakage): Partitioning Step



Full Reconstruction (with Rank Leakage): Proof Intuition

- Hard part is to show that O(N log N) queries suffice with a small constant.
- Proof consists of showing that **if** certain favourable queries are made, then partitioning succeeds in constructing N classes.
- Roughly speaking, for our proof we hope for queries on ranges:

  1. [x,*] for all $1 \le x \le N/2$ (left coupons)
  2. [*,y] for all $N/2+1 \le y \le N$ (right coupons)
  3. [N/2+1,y] and [x,N] for some $y \ge x$.

- Assuming these all arise, then a combinatorial argument establishes the success of the partitioning step.
- First two cases are essentially a pair of coupon collector problems – success with high probability with O(N log N) queries.
- Third case is a high probability event: $1 - e^{-Q/(2N+2)}$ for Q queries.

Full Reconstruction (**without** Rank Leakage)

- Can only recover values up to **reflection**.
- Data-optimal algorithm (fails ⇨ full reconstruction is impossible).
- Expected number of sufficient queries is at most

  $N \cdot (3 + \log N)$ for $N \geq 26$

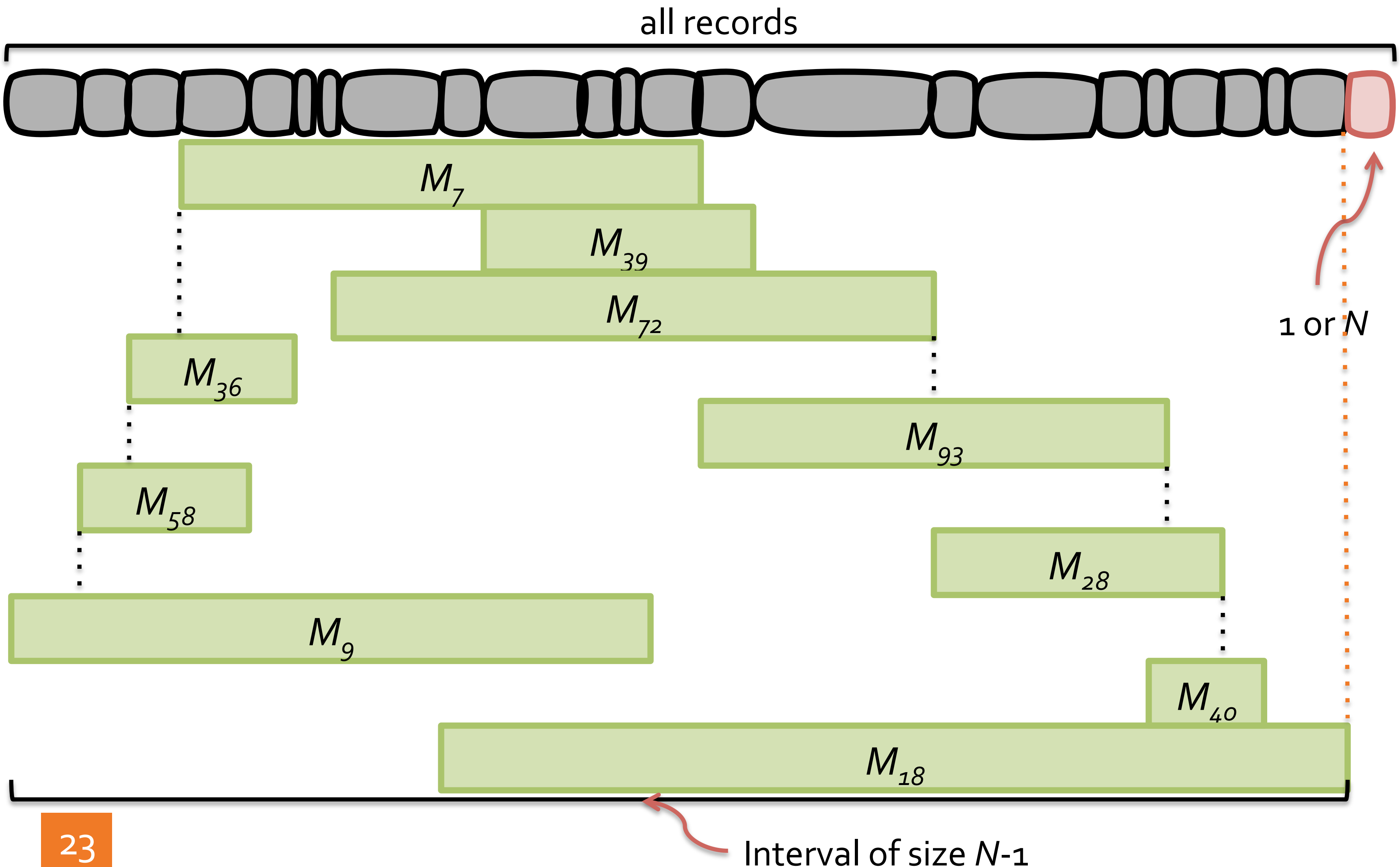
- Partition (as before), then sort*.
- Expected number of necessary queries is at least

  $1/2 \cdot N \cdot \log N - O(N)$

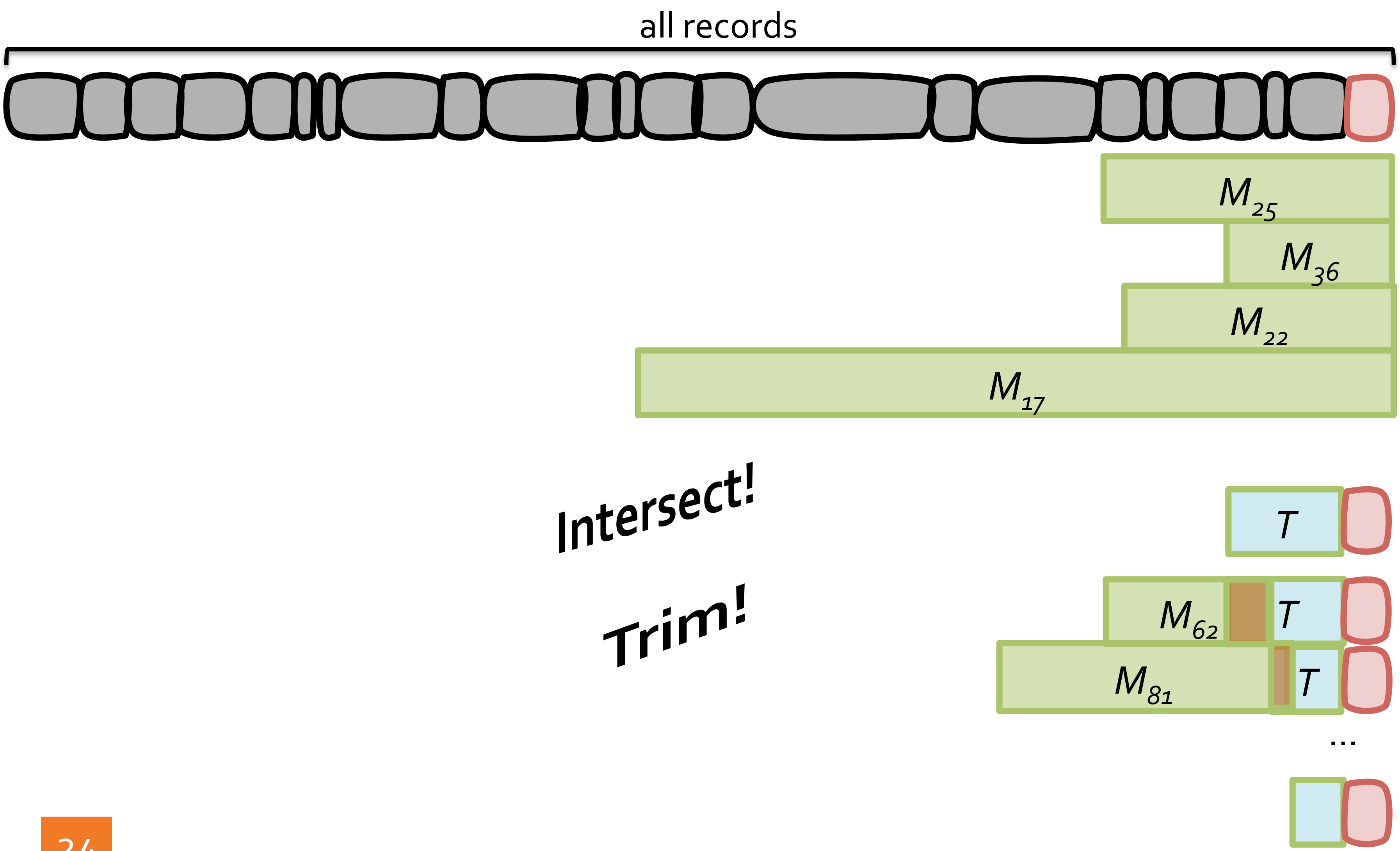
  - for any algorithm.

*Not quite.

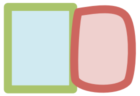
Full Reconstruction (without Rank Leakage): Sorting Step



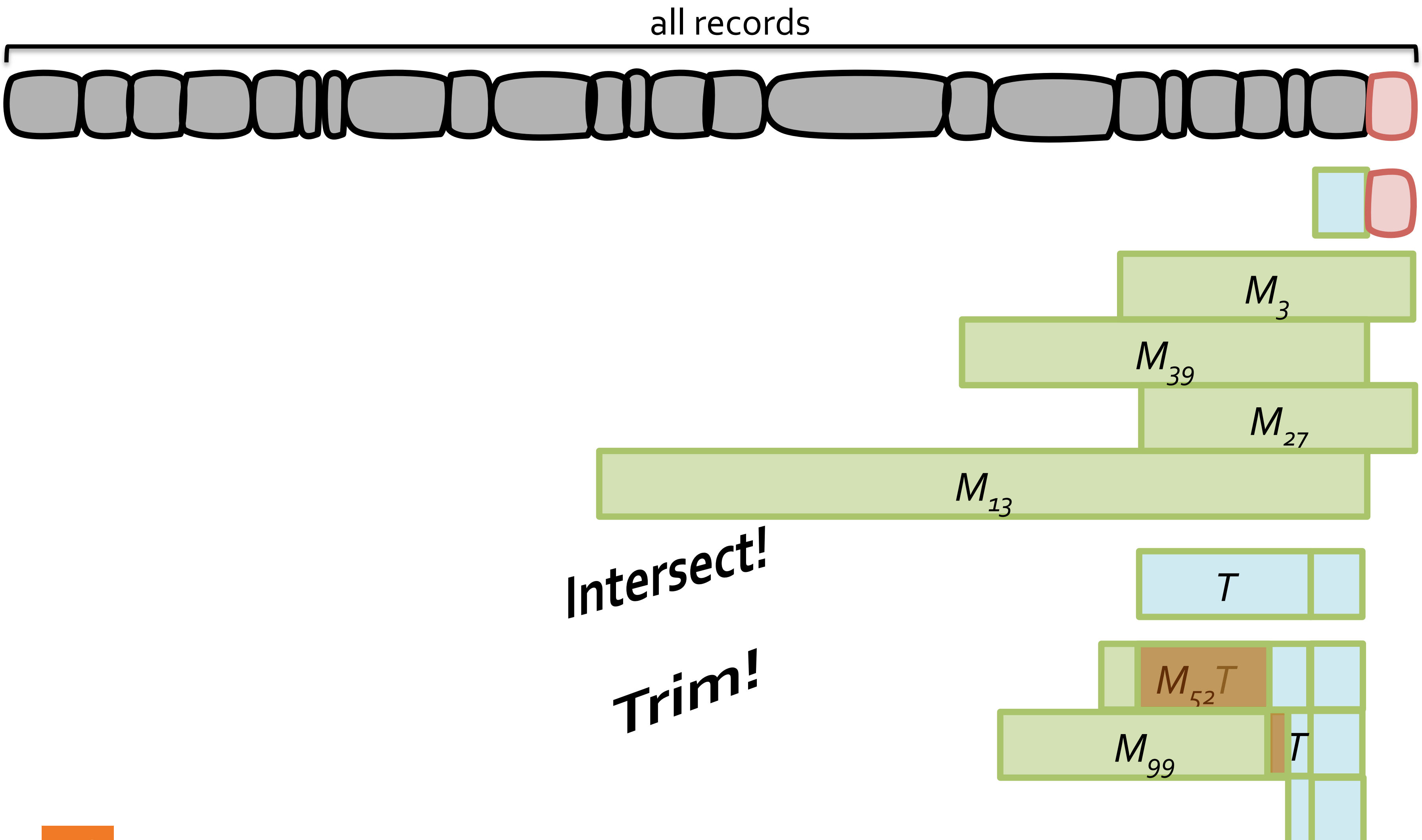
Full Reconstruction (without Rank Leakage): Sorting Step – Extending



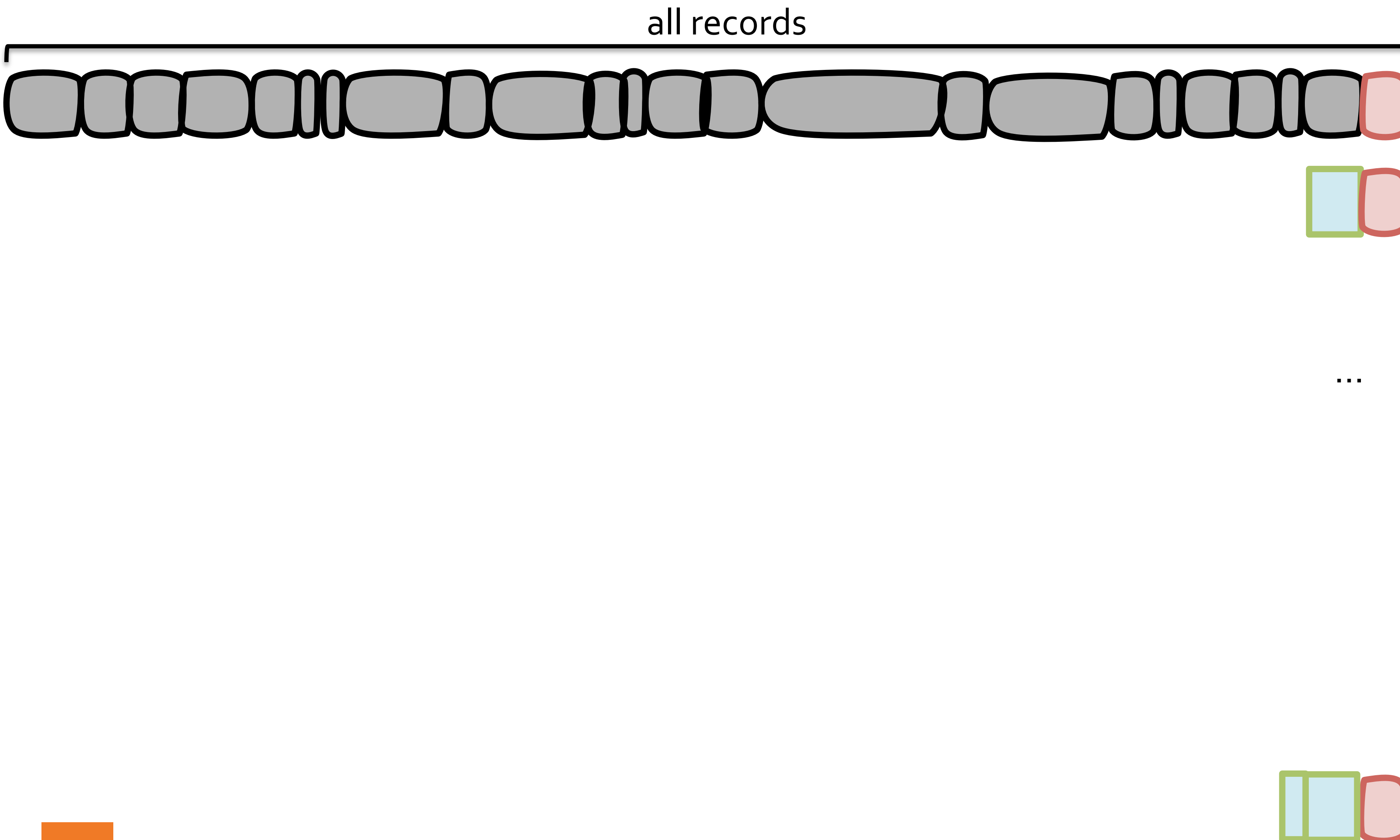
Full Reconstruction (without Rank Leakage): Sorting Step – Extending



Full Reconstruction (without Rank Leakage): Sorting Step



Full Reconstruction (without Rank Leakage): Sorting Step



Full Reconstruction (without Rank Leakage): Proof Intuition

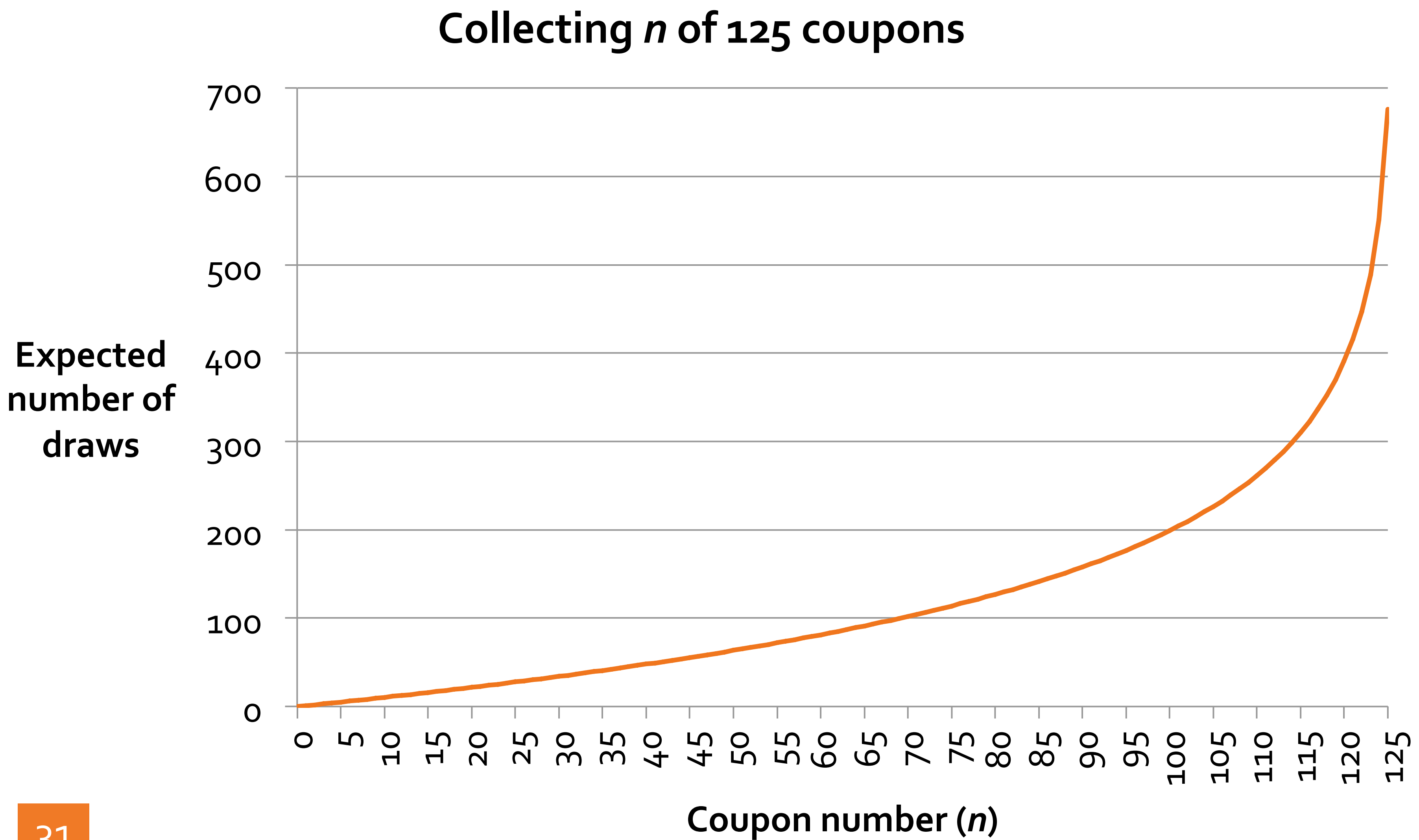
- Hard part is again to show that O(N log N) queries suffice, with a small constant.
- Proof again consists of showing that **if** certain favourable queries are made, then partitioning succeeds in constructing N classes.
- Coupon collecting bounds then establish that O(N log N) queries are enough.

Attack 2: Approximate Reconstruction

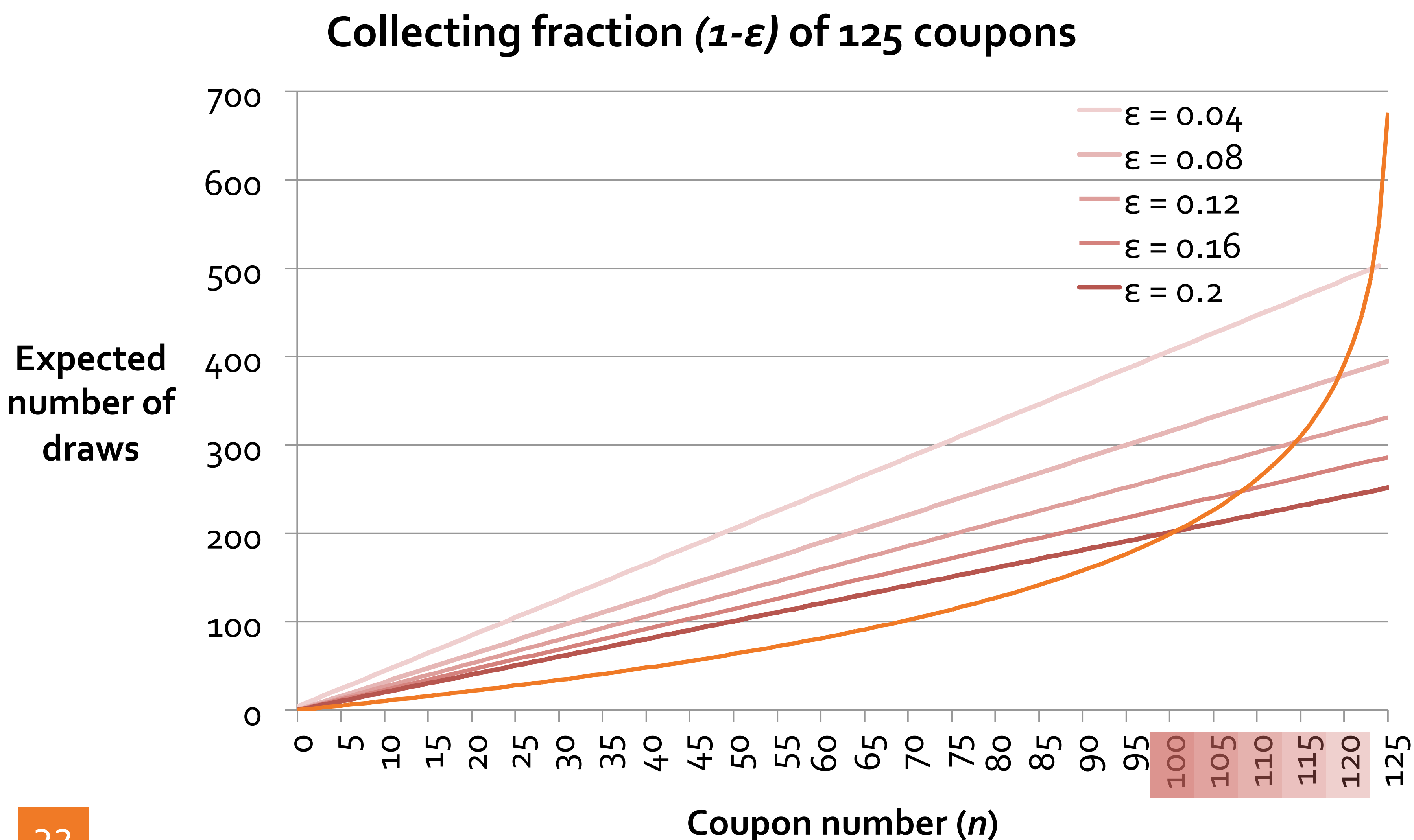
Approximate Reconstruction Attack (without Rank Leakage)

- Recover values up to **reflection** and with relative error ε.
- Expected number of sufficient queries is
  $$5/4 \cdot N \cdot (\log 1/\varepsilon) + O(N).$$
- Expected number of necessary queries is at least
  $$1/2 \cdot N \cdot (\log 1/\varepsilon) - O(N)$$
  for any algorithm.
- Not data-optimal without rank leakage (but *is* with it)

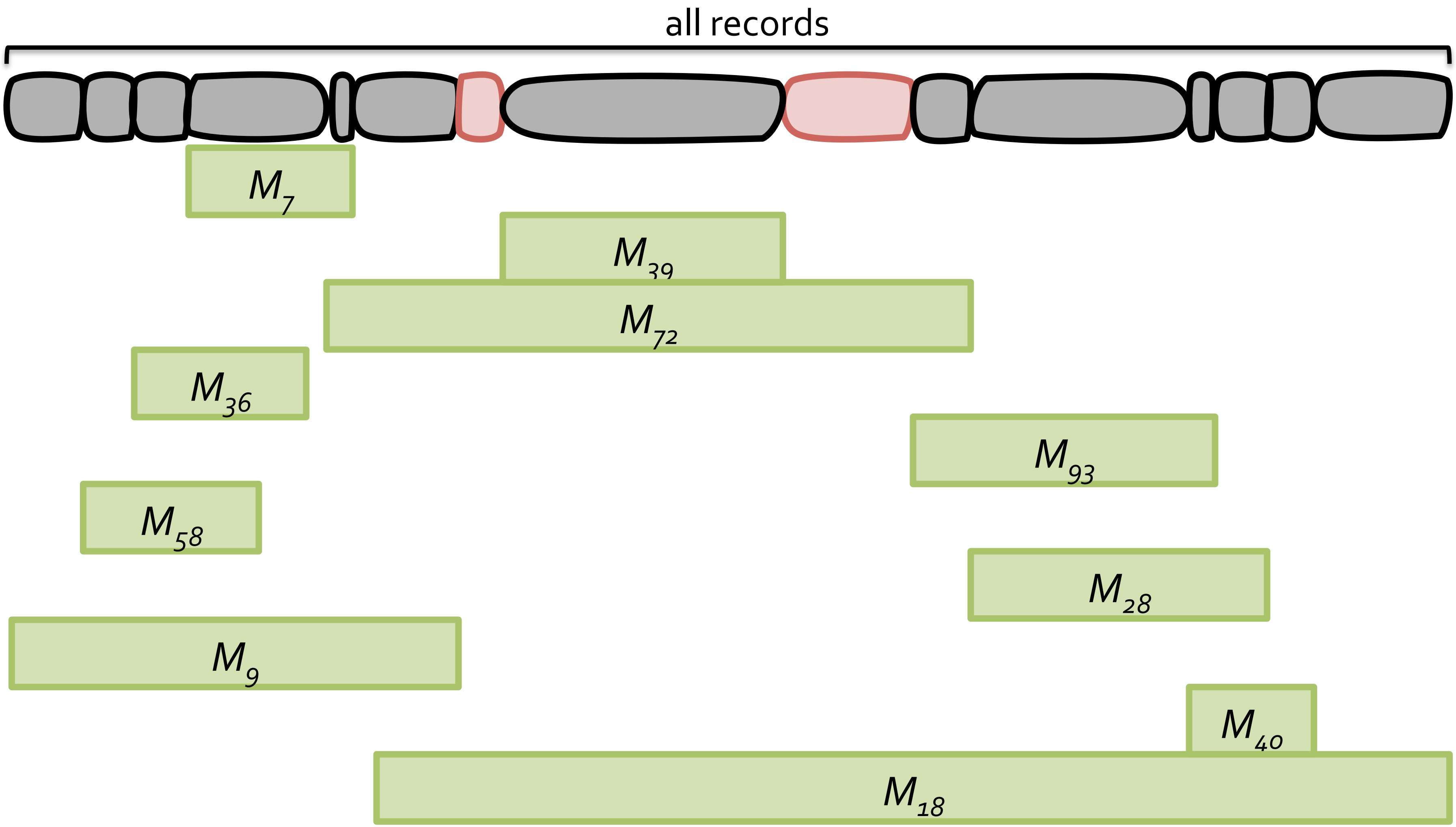
Coupon Collection (N=125)



Coupon Collection (N=125)

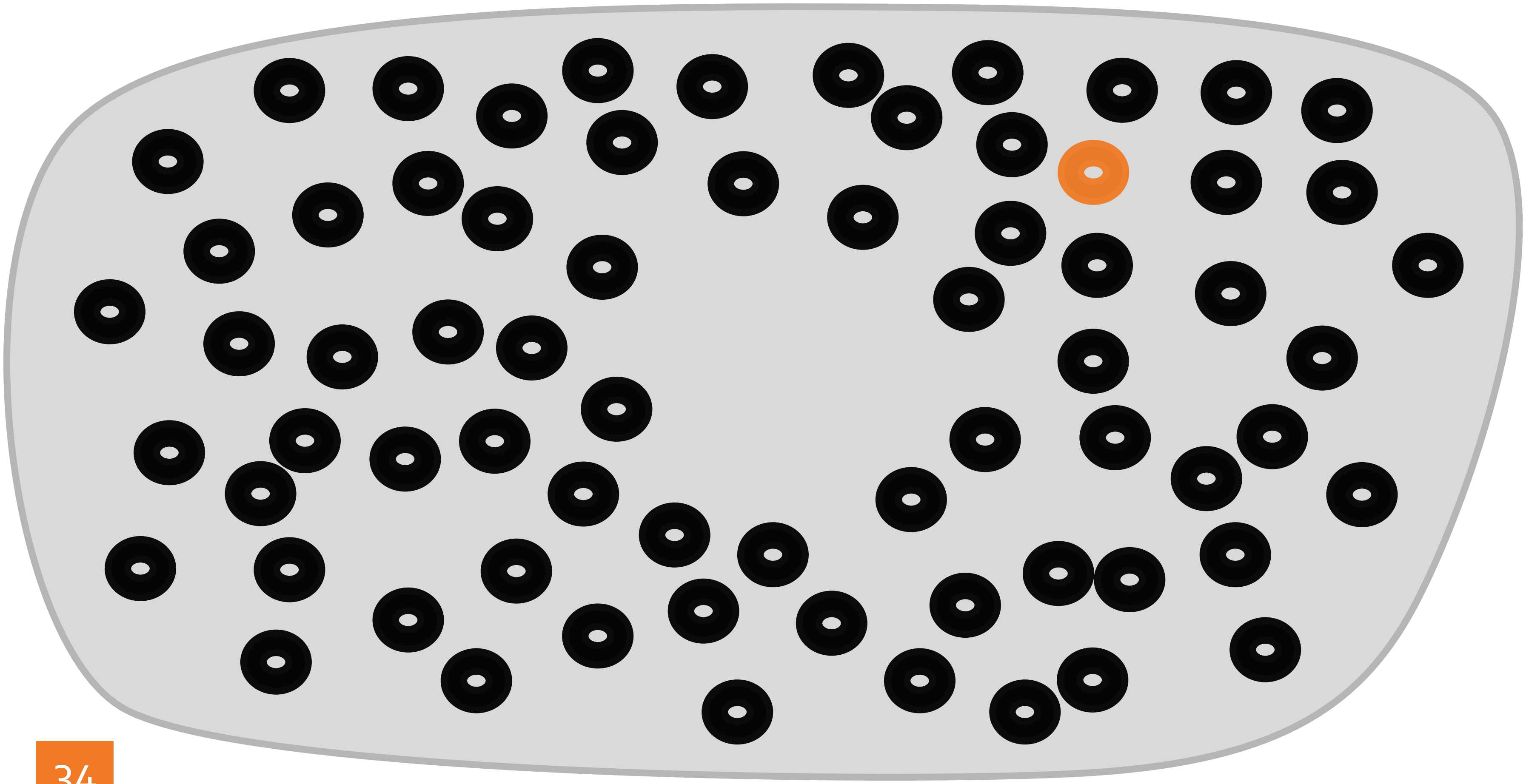


Approximate Reconstruction: Old Partitioning Method Doesn't Work



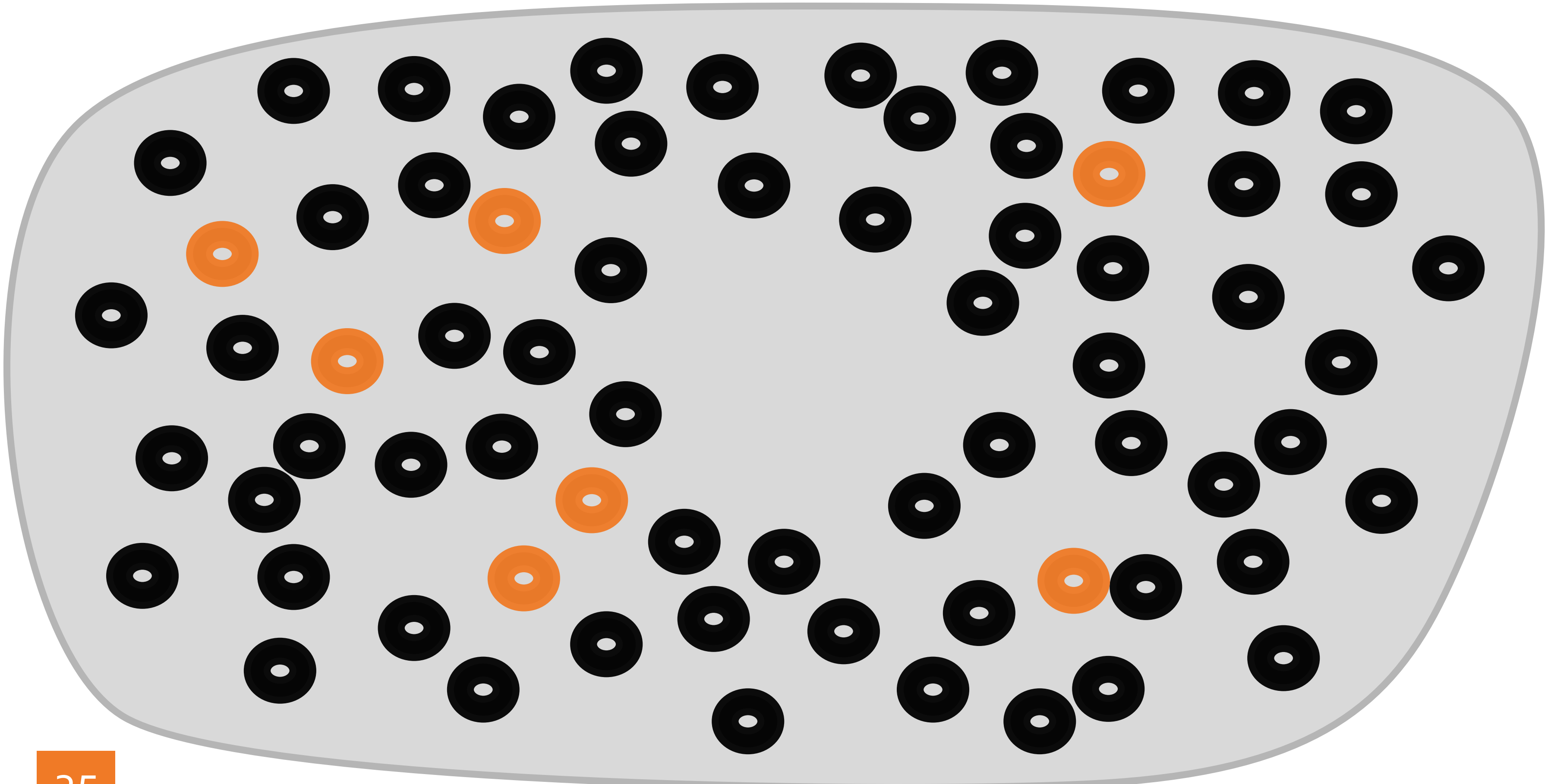
Approximate Reconstruction: Partitioning Step

1. Pick any record *r*.



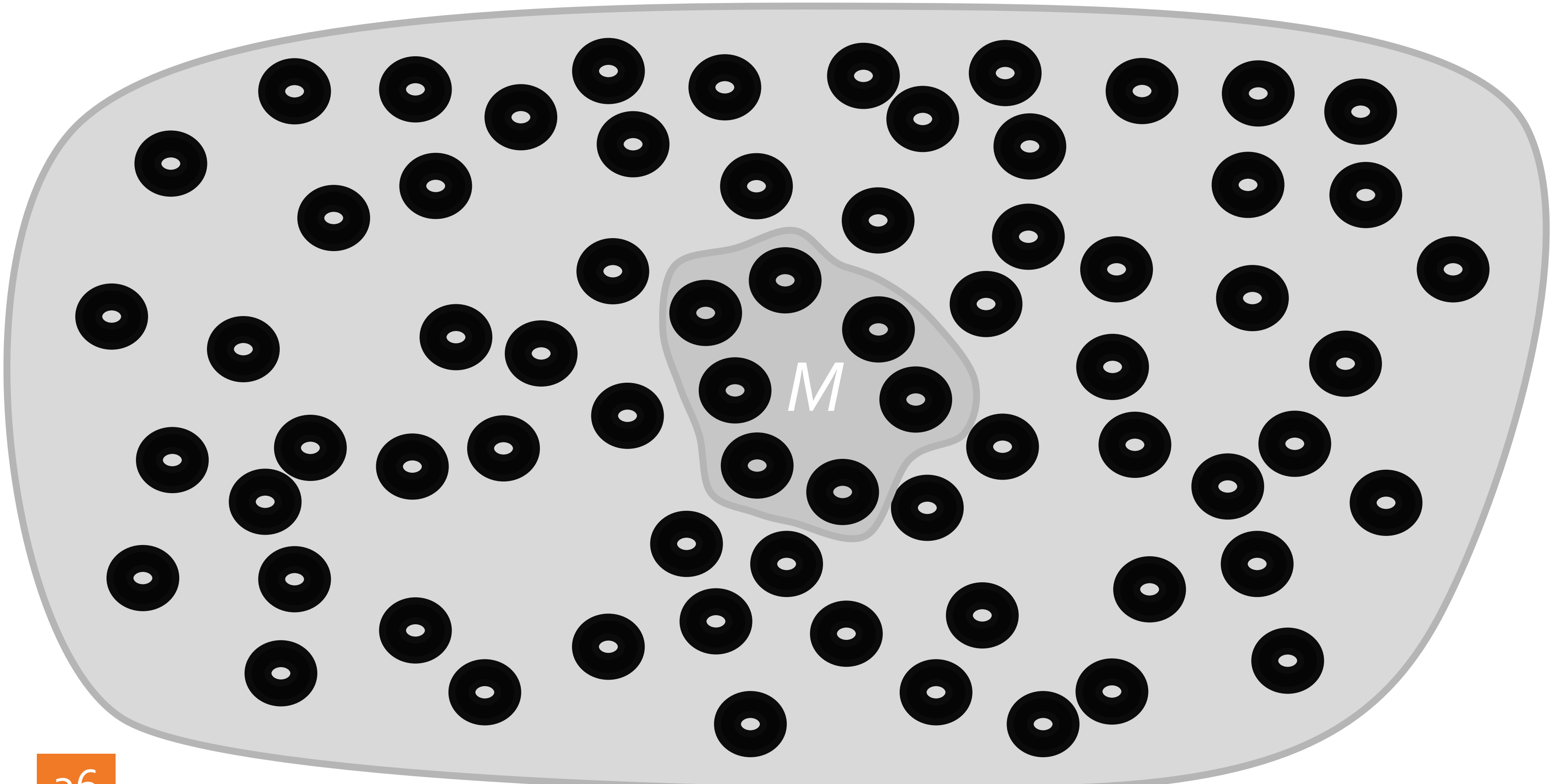
Approximate Reconstruction: Partitioning Step

2. Intersect all queries matching *r* to get ***M***.



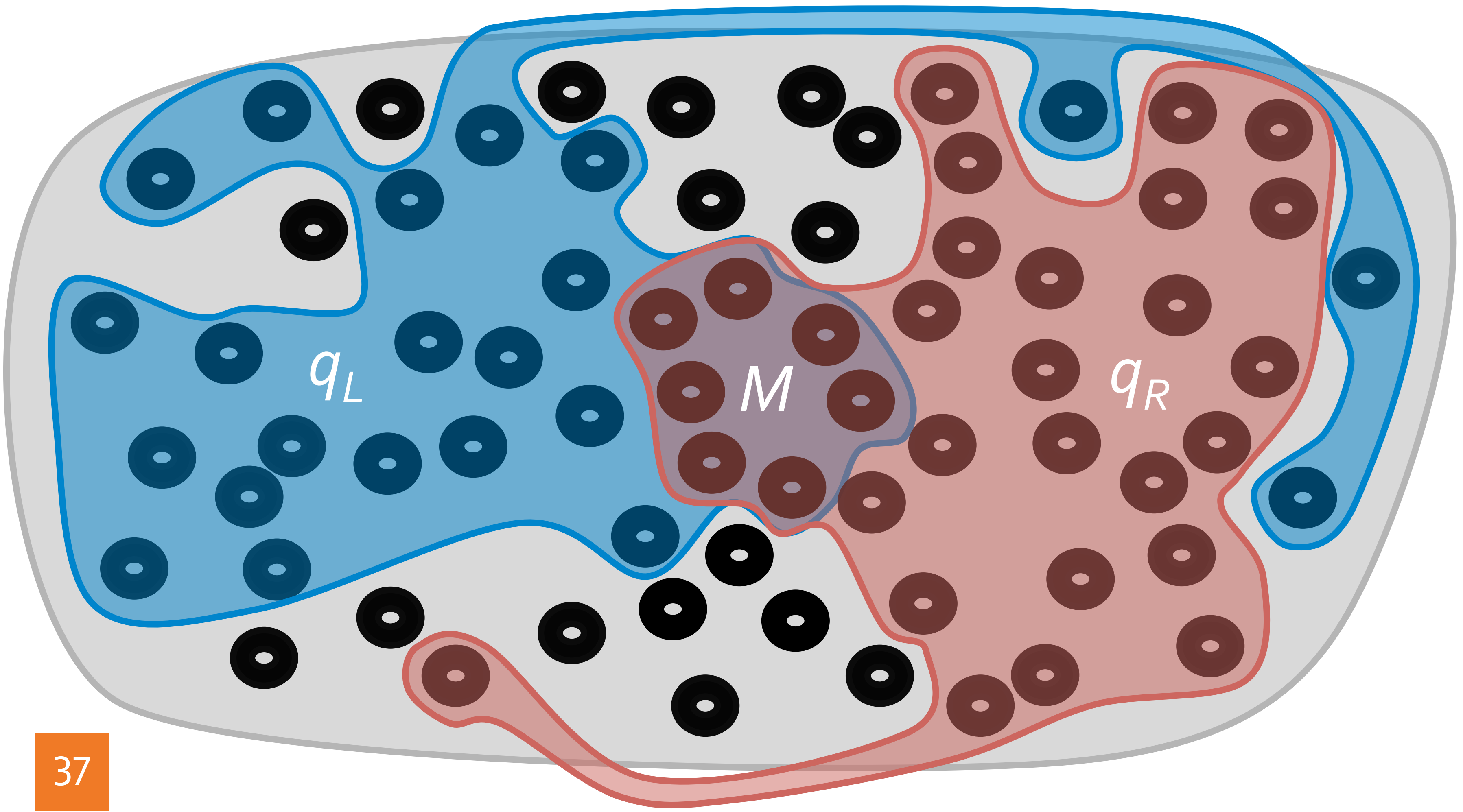
Approximate Reconstruction: Partitioning Step

2. Intersect all queries matching *r* to get ***M***.



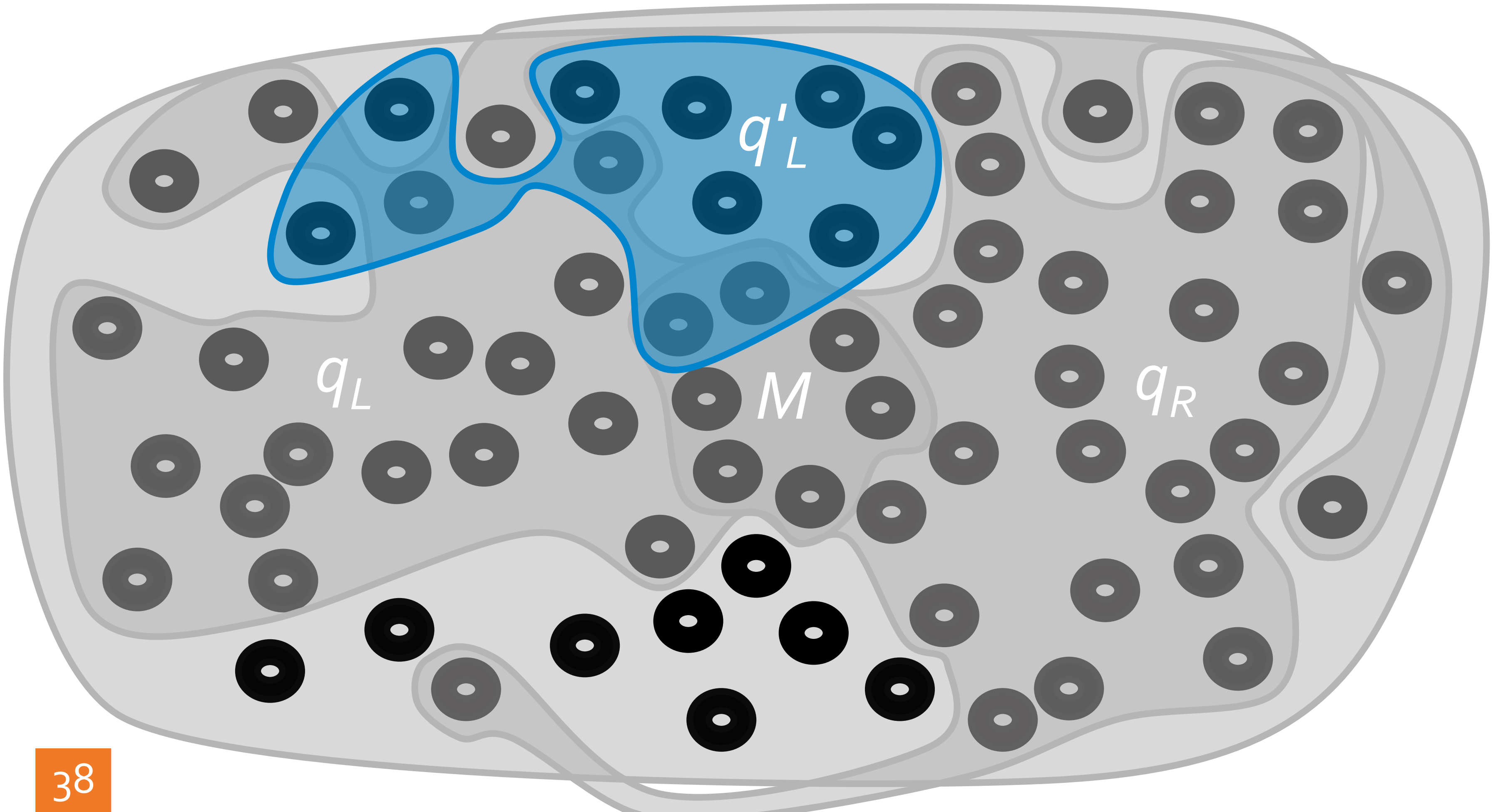
Approximate Reconstruction: Partitioning Step

3. Find q_L and q_R : $q_L \cap q_R = M$ and $|q_L \cup q_R|$ maximised.



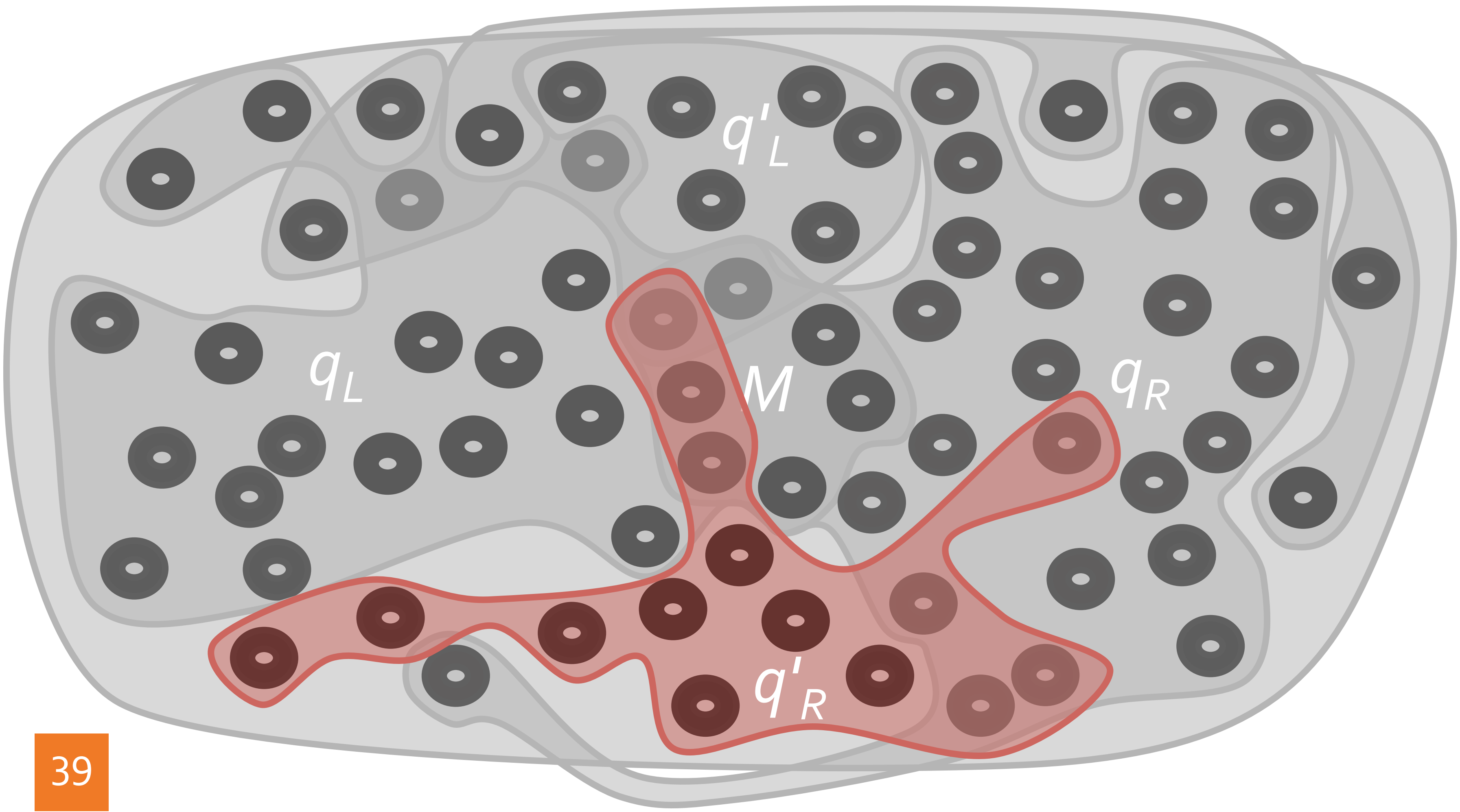
Approximate Reconstruction: Partitioning Step

4. Find $q'_L : q_L \cap q'_L \neq \emptyset$, $q'_L \cap q_R \subseteq M$, $|q_L \cup q'_L|$ maximised.



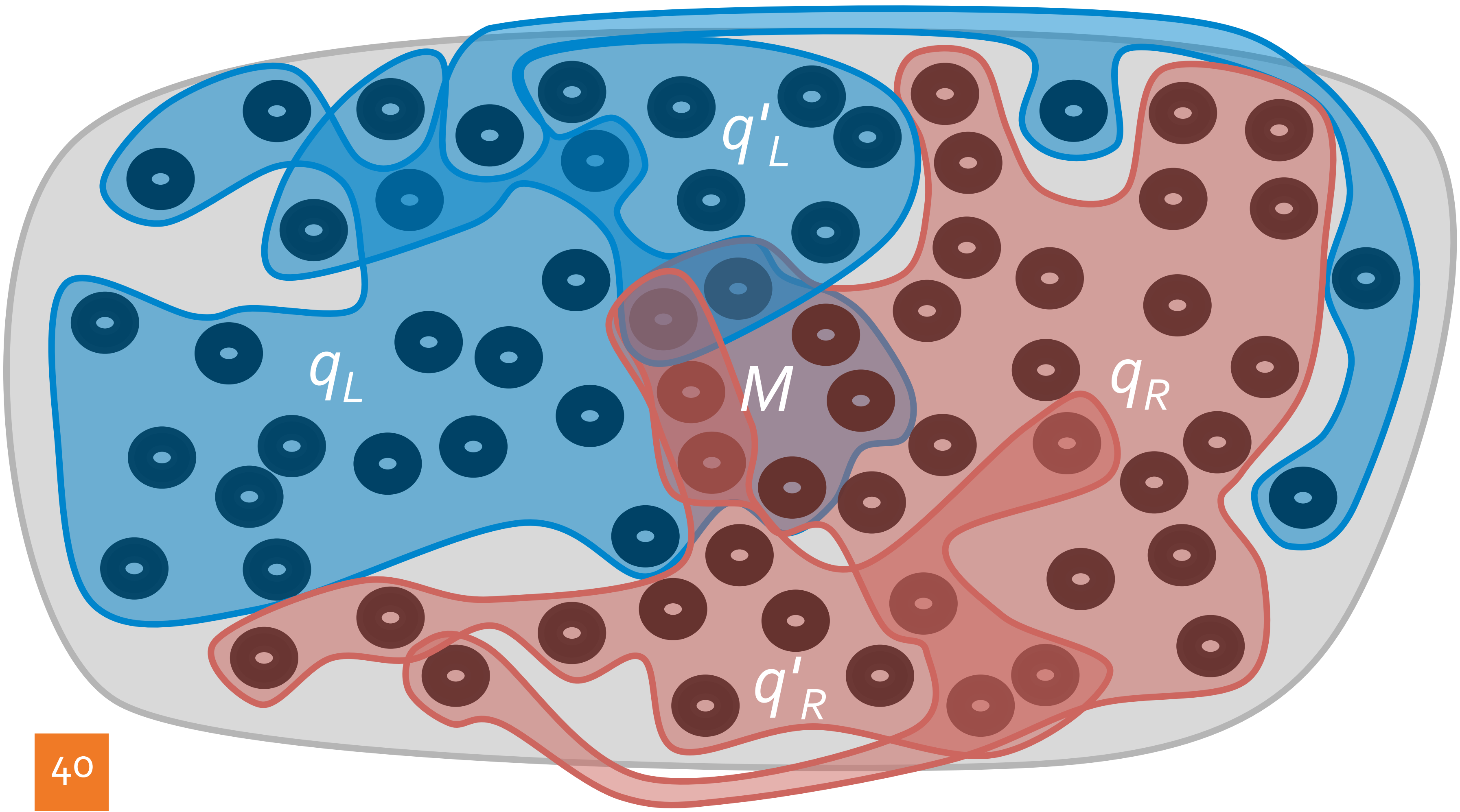
Approximate Reconstruction: Partitioning Step

5. Find $q'_R : q_R \cap q'_R \neq \varnothing$, $q'_R \cap q_L \subseteq M$, $|q_R \cup q'_R|$ maximised.



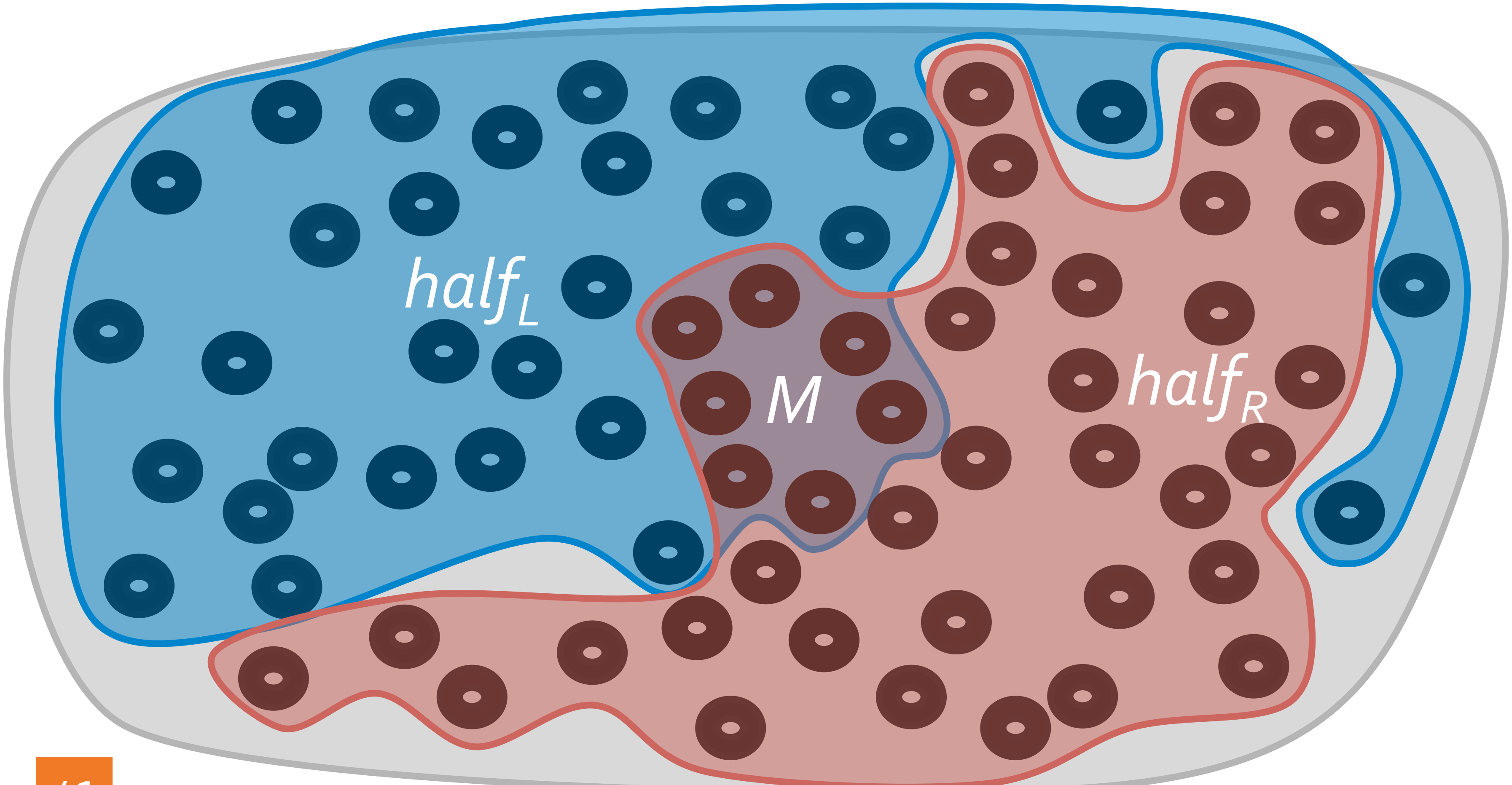
Approximate Reconstruction: Partitioning Step

6. Start over if not every record is in $\boldsymbol{q_L} \cup \boldsymbol{q'_L} \cup \boldsymbol{q_R} \cup \boldsymbol{q'_R}$.



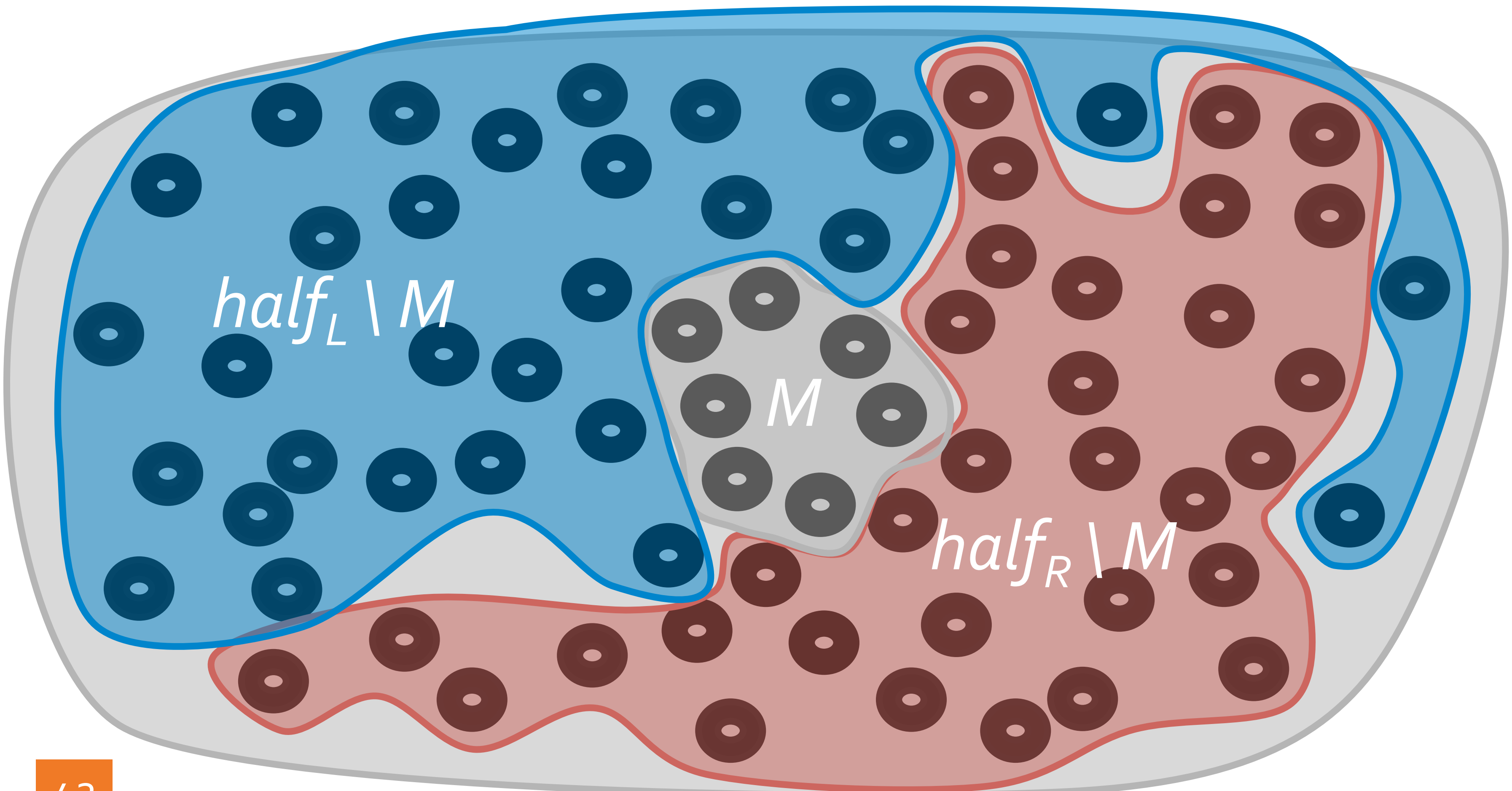
Approximate Reconstruction: Partitioning Step

7. Split into ***half**$_L$* = ***q**$_L$* ∪ ***q'**$_{L}$*, ***half**$_R$* = ***q**$_L$* ∪ ***q'**$_{L}$*, and ***M***.



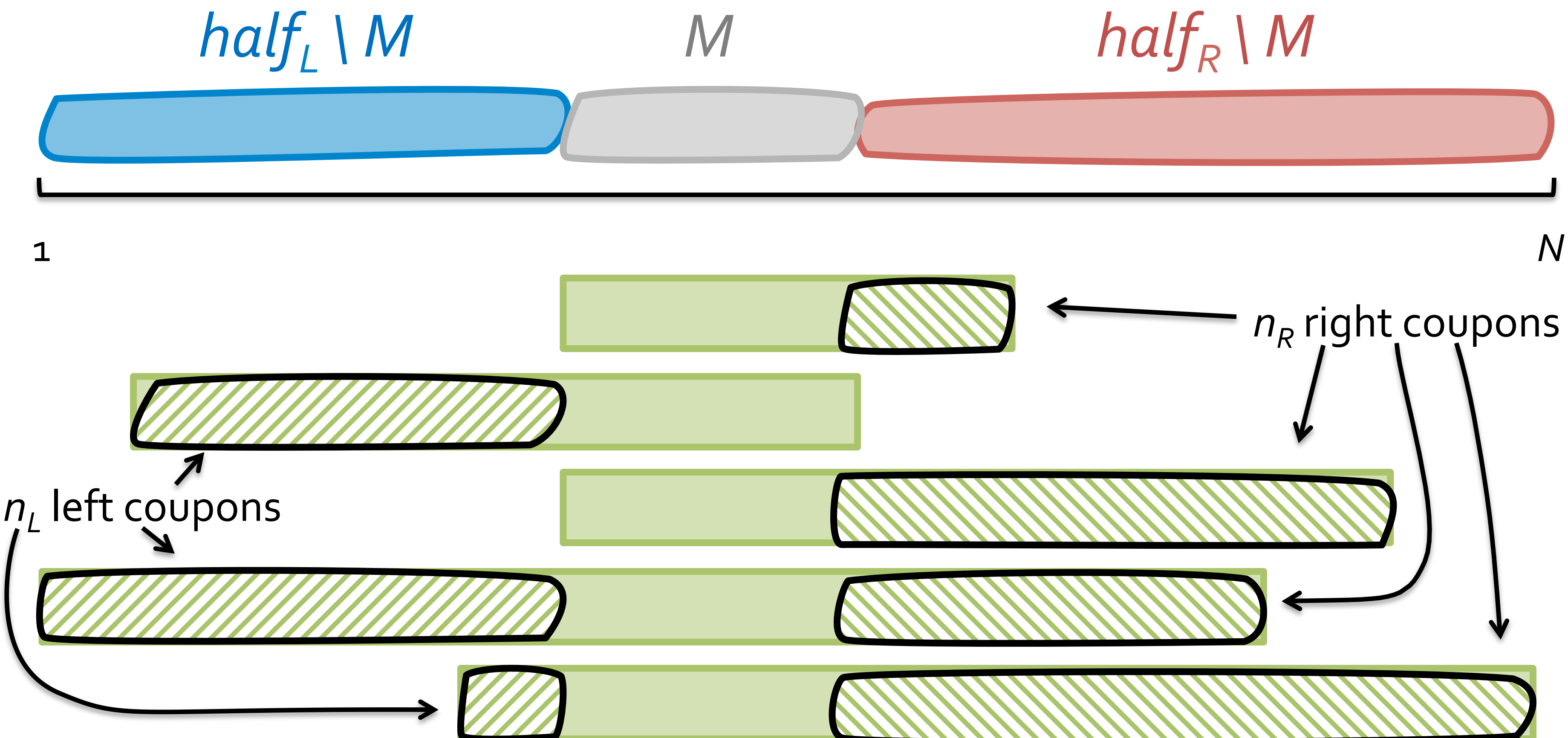
Approximate Reconstruction: Partitioning Step

7. Split into $half_L = q_L \cup q'_L$, $half_R = q_L \cup q'_L$, and M.



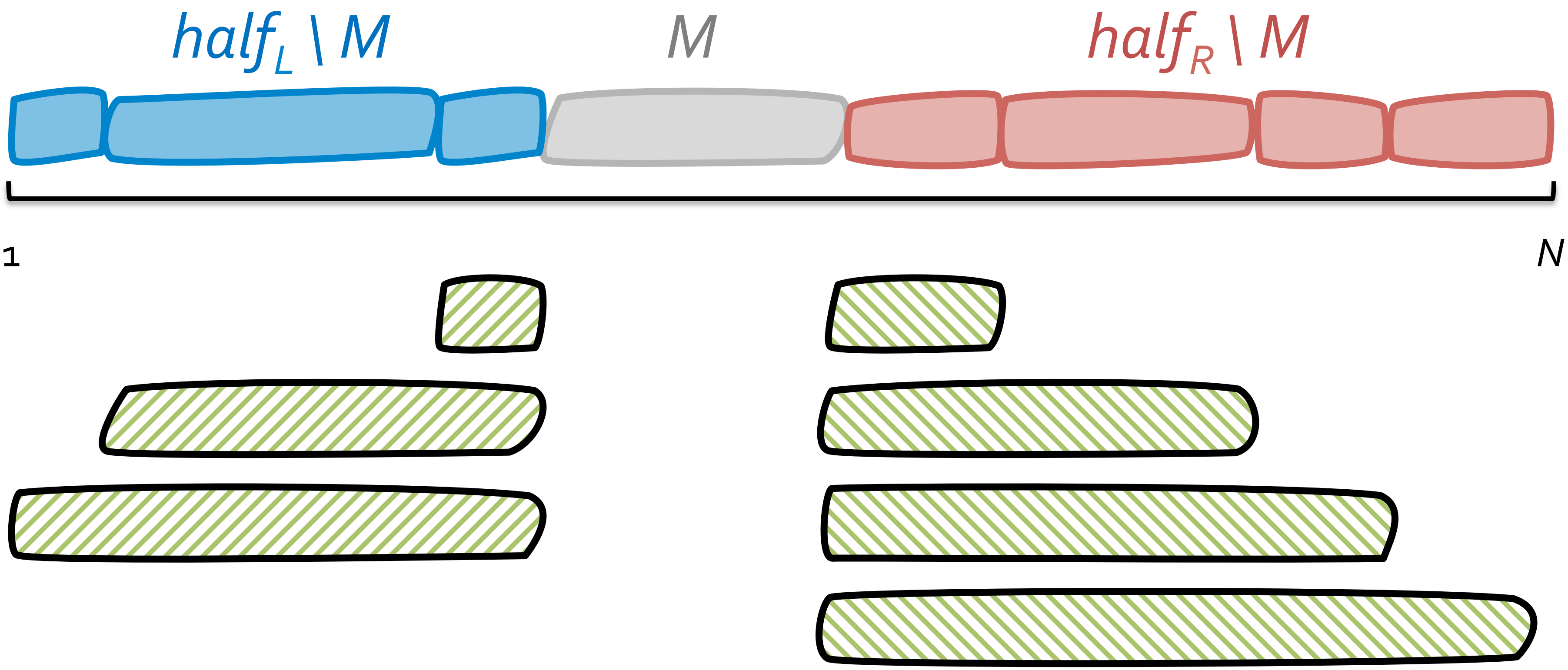
Approximate Reconstruction: Sorting Step

8. Form **left & right coupons** with queries containing ***M***.



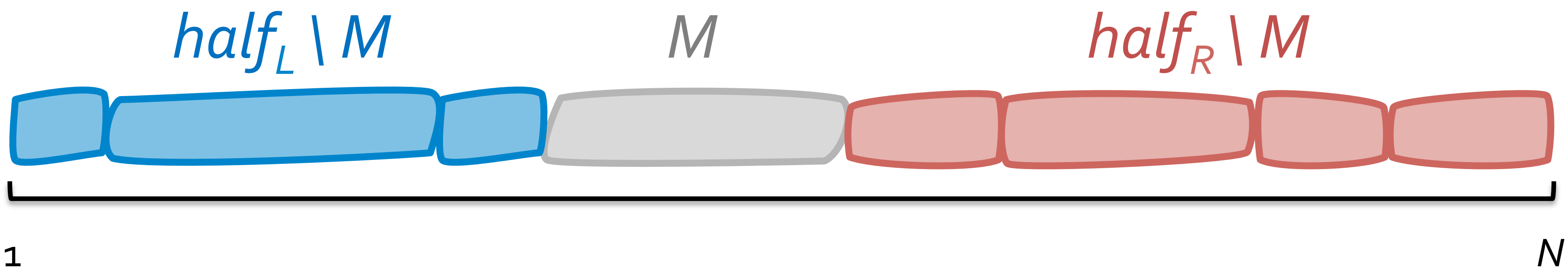
Approximate Reconstruction: Sorting Step

9. Use left & right coupons to **sort** ***half$_L$* \ *M* & *half$_R$* \ *M***.



Approximate Reconstruction: Sorting Step

9. Use left & right coupons to **sort** $\boldsymbol{half_L \setminus M}$ **&** $\boldsymbol{half_R \setminus M}$.



$$n_L + 1 + n_R = (1\text{-}\varepsilon) \cdot N$$

$$\Downarrow$$

reconstruction with precision $\varepsilon \cdot N$

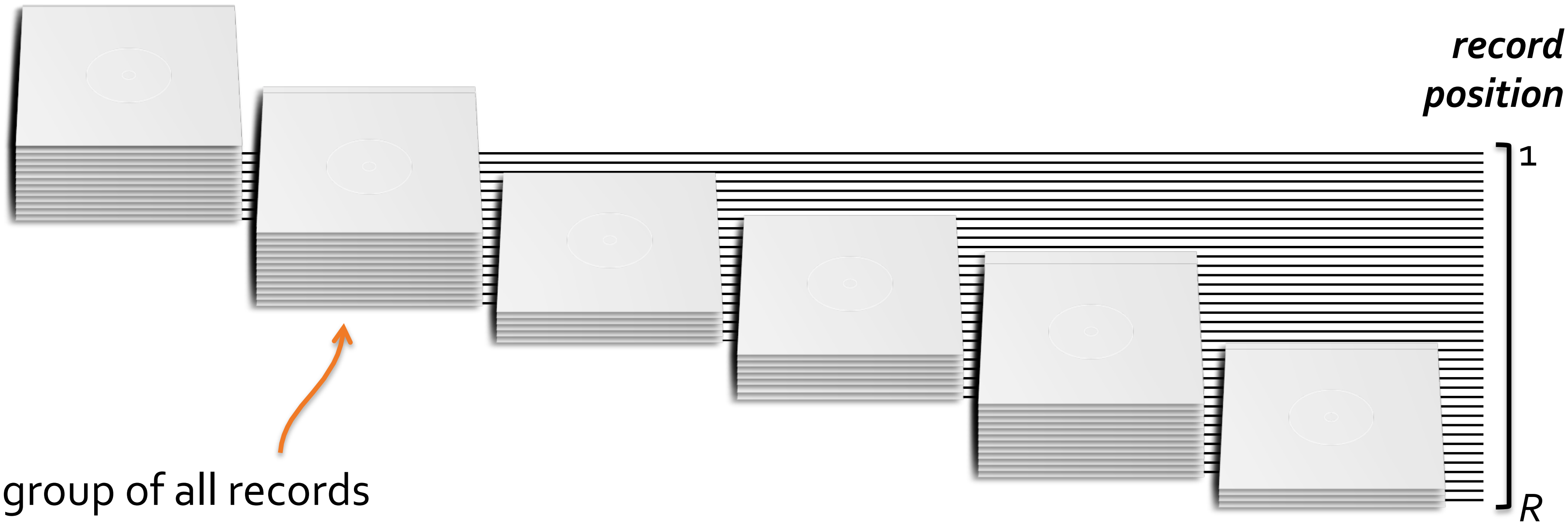
Attack 3: Reconstruction with Auxiliary Data

Reconstruction with Auxiliary Data and Rank Leakage

- As before, queries have **ranges** chosen **uniformly at random**.
- Assume access pattern and rank are leaked.
- We now also assume that an **approximation to the distribution on values** is known.
  - "Auxiliary data".
  - From aggregate data, or from another reference source.
- We show experimentally that, under these assumptions, far fewer queries are needed.
- Now no requirement on density, so interesting for OPE and ORE schemes too (OPE/ORE schemes are trivial to break in dense case).

Auxiliary Data Attack: Partitioning Step

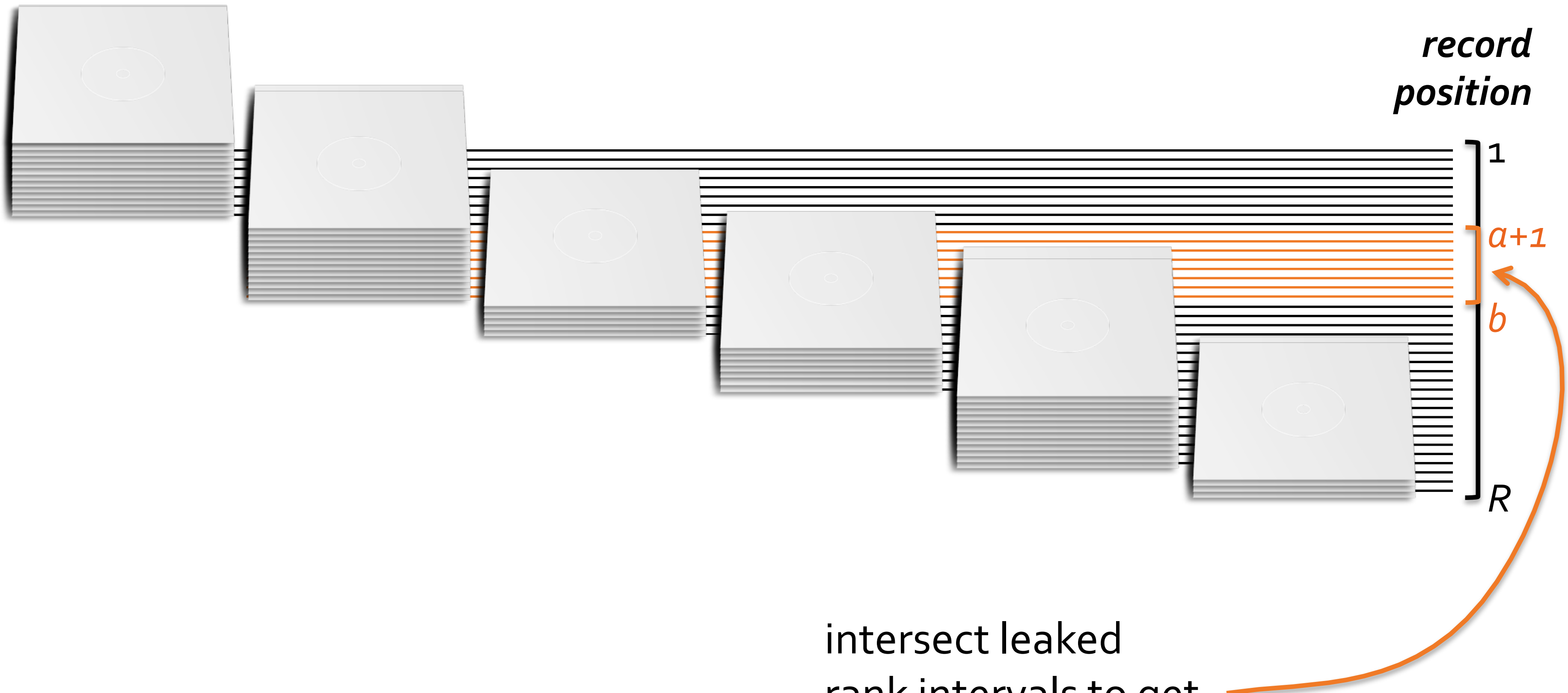
1. **Partition** records as in full reconstruction attack.



group of all records appearing in exact same subset of queries

Auxiliary Data Attack: Partitioning Step

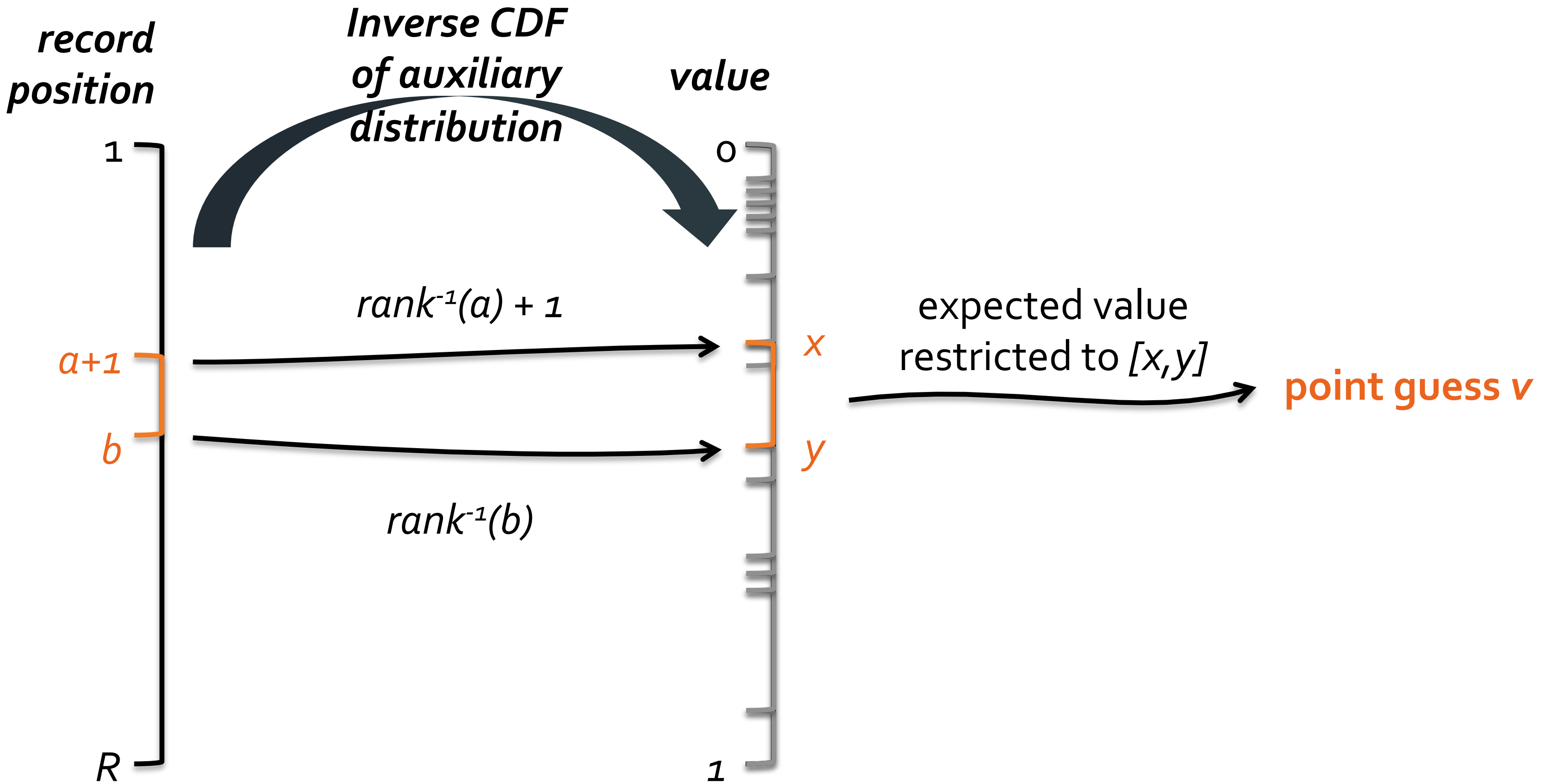
2. Assign a **position interval** to each partition.



intersect leaked rank intervals to get **position interval**

Auxiliary Data Attack: Estimating Step

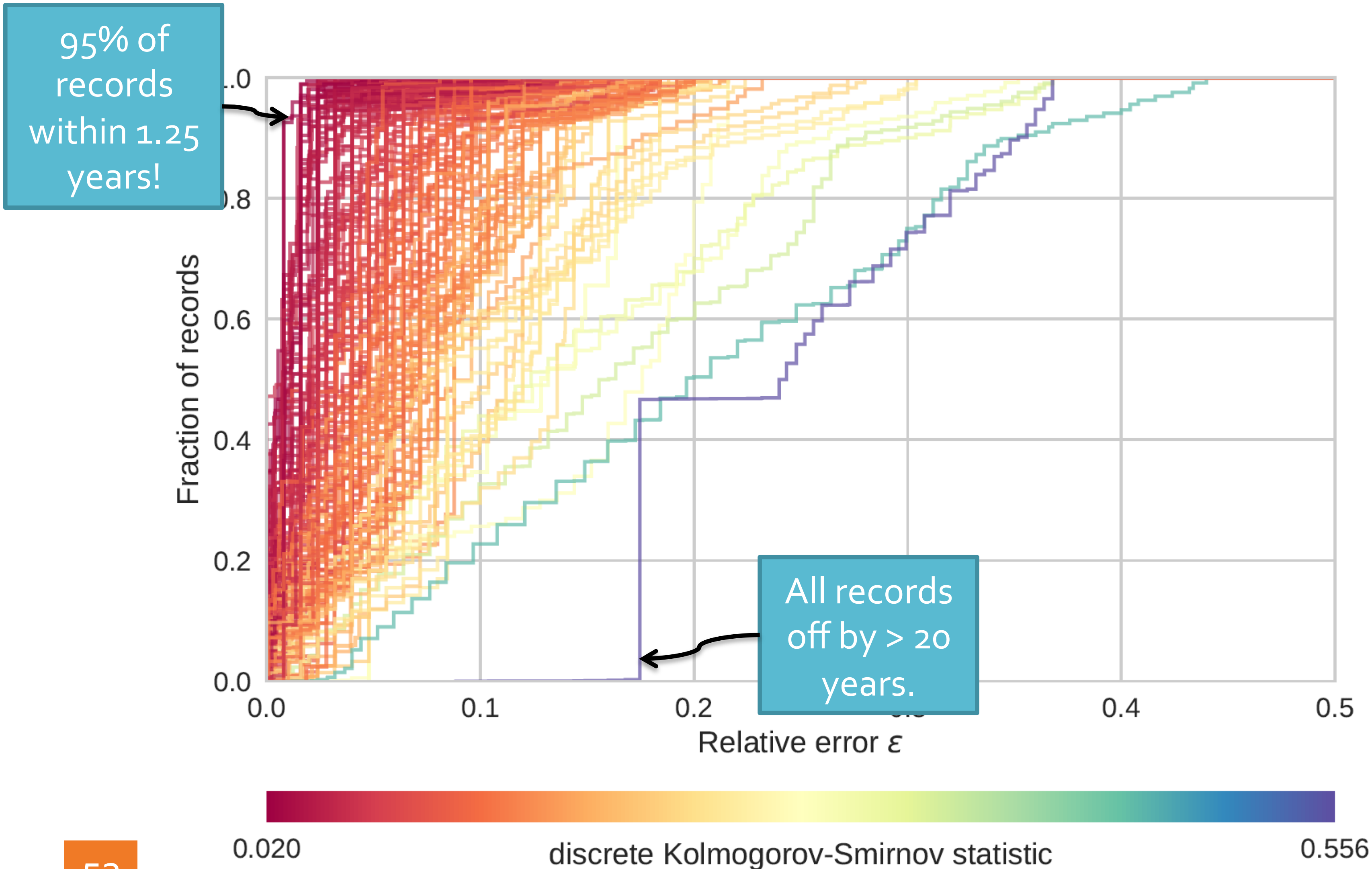
3. Assign a **value** to each group's position interval



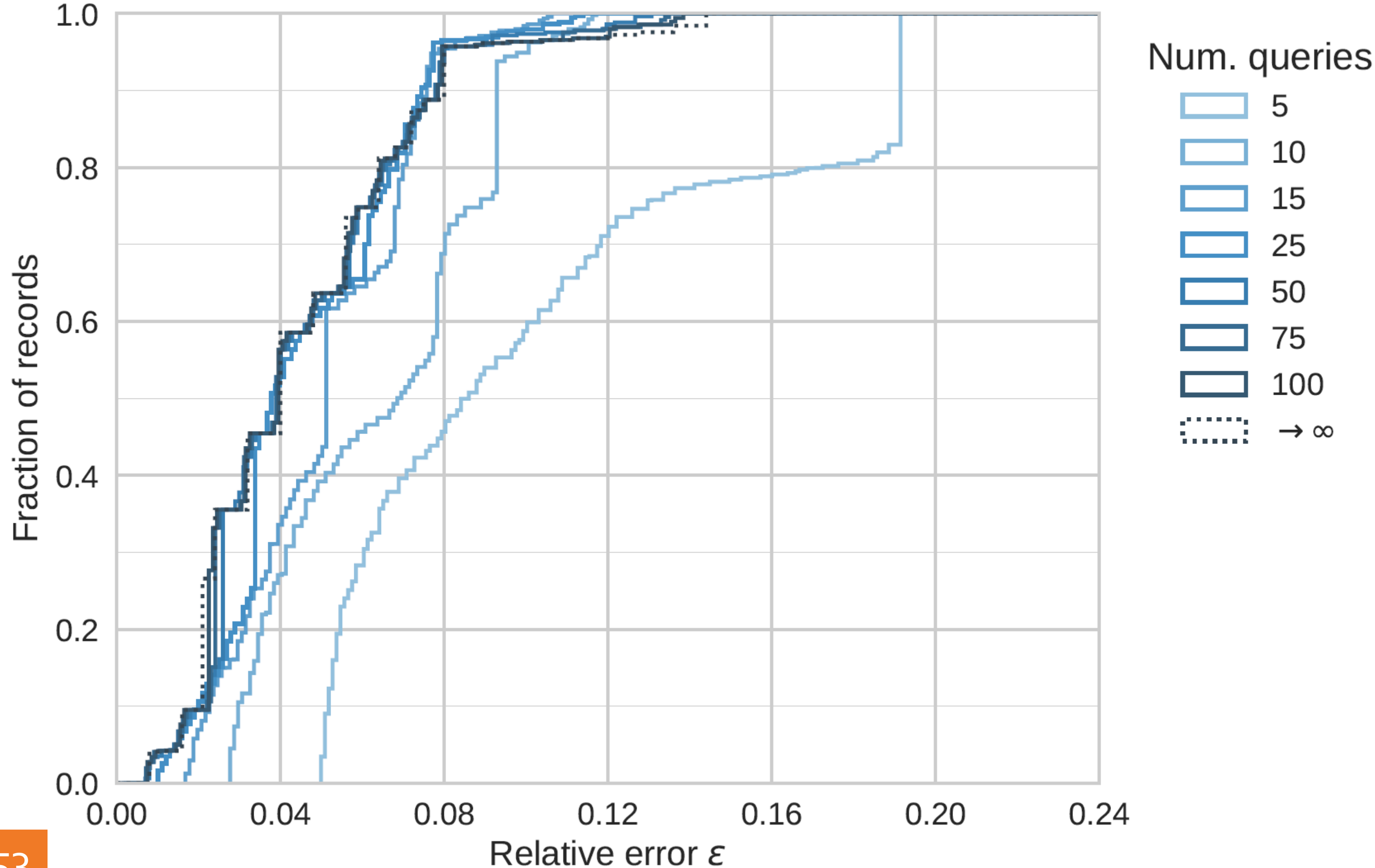
Auxiliary Data Attack: Experimental Evaluation

- Ages, $N = 125$ (0 to 124).
- Health records from US hospitals (NIS HCUP 2009).
- Target data: individual hospitals' records.
- Auxiliary data: aggregate of 200 hospitals' records.
- Measure of success: proportion of records with value guessed within ε.

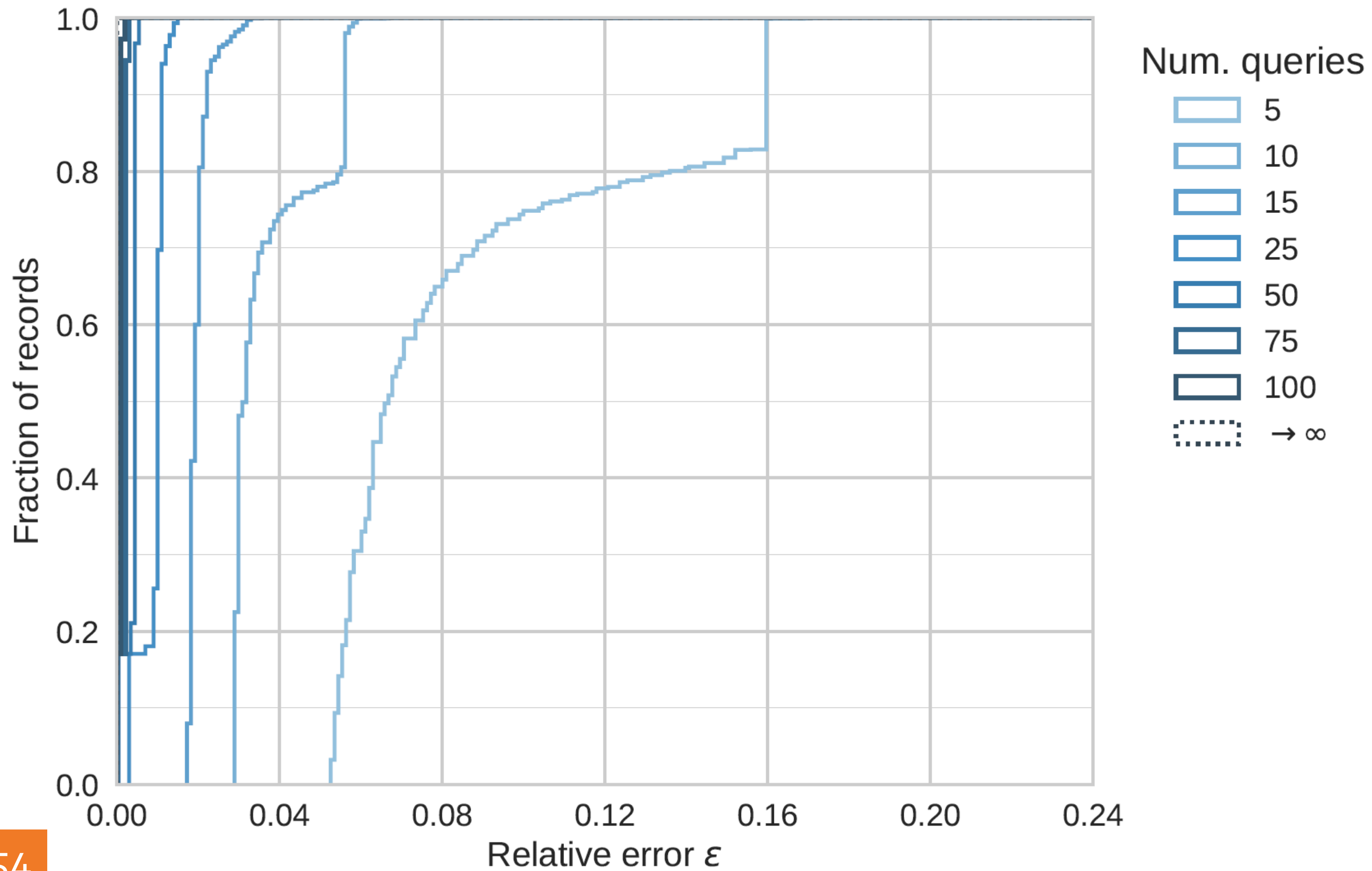
Auxiliary Data Attack: Asymptotic Success Rates for Different Target Hospitals



Auxiliary Data Attack: Results for Typical Target Hospital

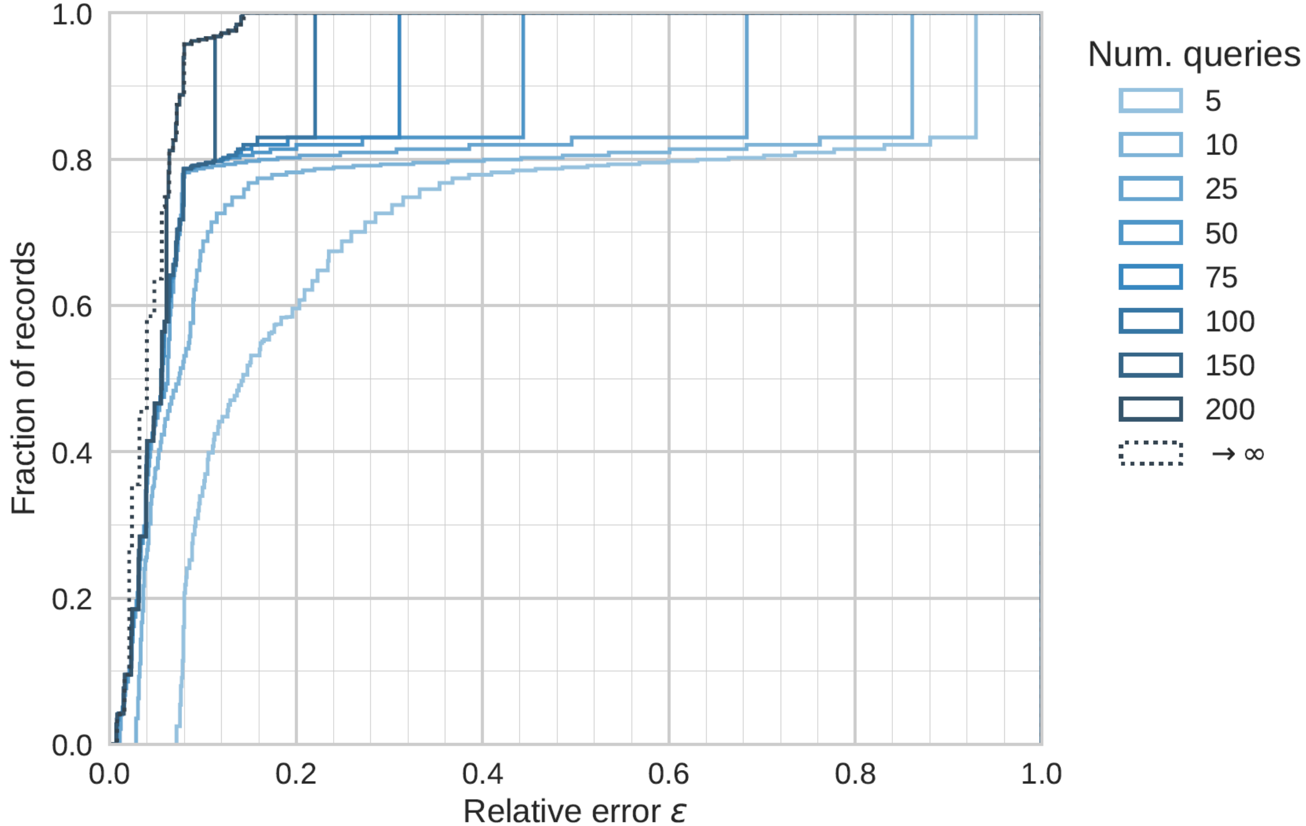


Auxiliary Data Attack: Results with Perfect Auxiliary Distribution



Auxiliary Data Attack: Removing Assumptions

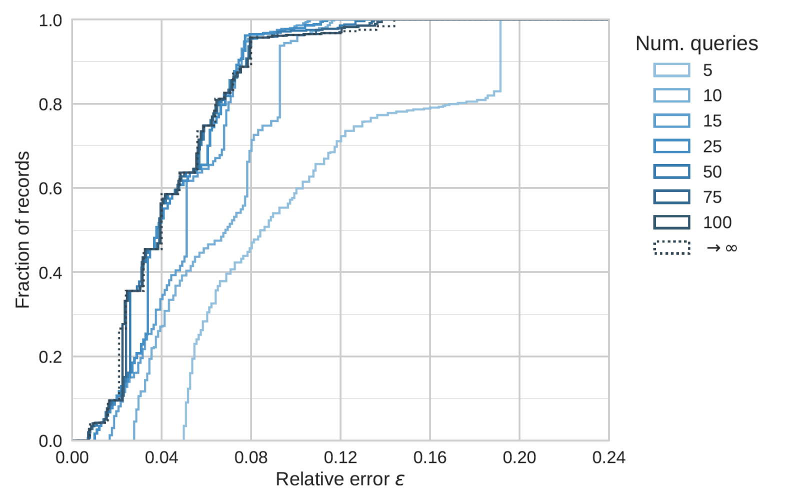
- Estimating total number of records is **fast** if not known *a priori*
- Learning set of record identifiers **can be slow** if not known *a priori:*

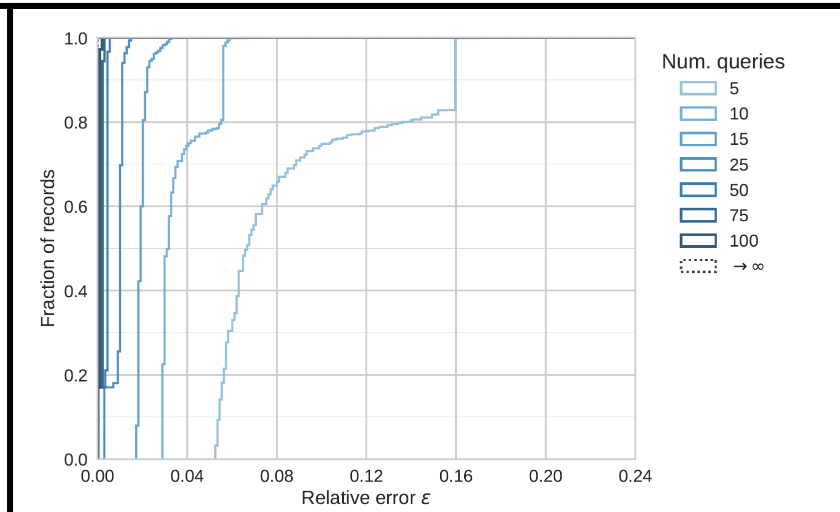


Auxiliary Data Attack: Removing Assumptions

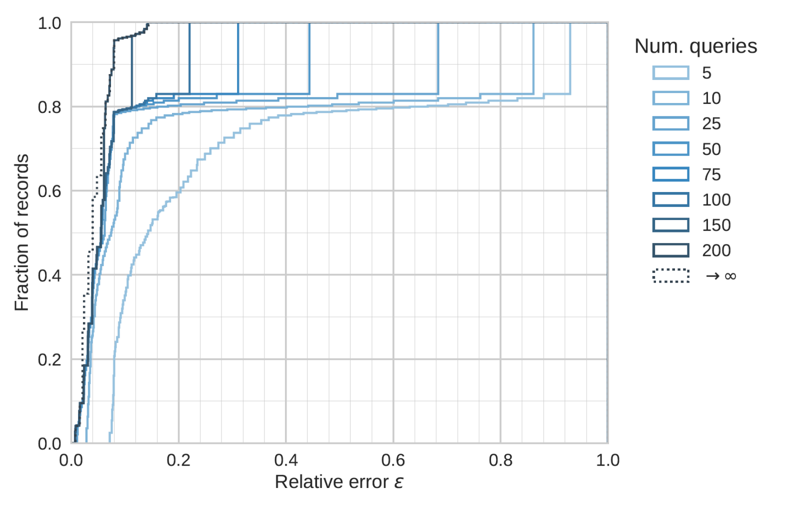
approximate auxiliary info. | **exact auxiliary info.**

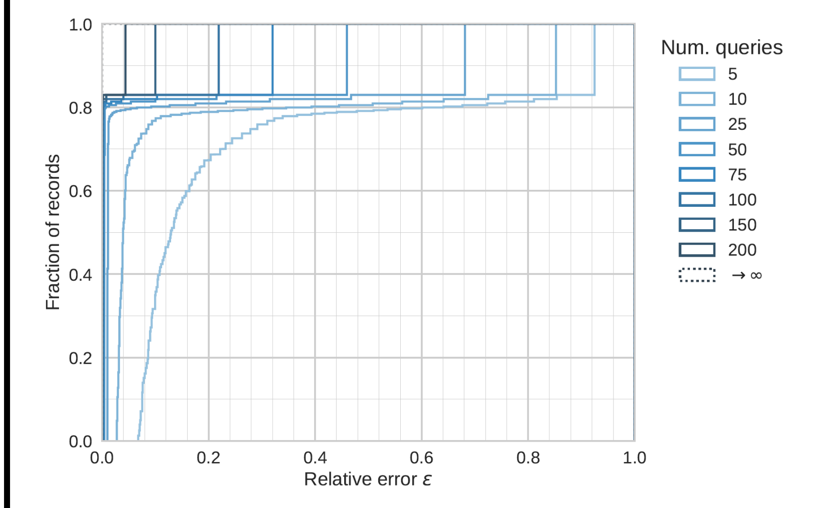
known record IDs





unknown record IDs





Summary and Conclusions

Summary of Our Attacks

Attack	Req'd leakage	Other req'ts	Suff. # queries
Full	AP + rank	Density	$N \cdot (\log N + 2)$
	AP	Density	$N \cdot (\log N + 3)$
ε-approximate	AP	Density	$5/4\, N \cdot (\log 1/\varepsilon) + O(N)$
Auxiliary	AP + rank	Auxiliary dist.	???

Conclusions

- Many clever schemes have been designed, enabling range queries on encrypted data:
  - OPE, ORE schemes.
  - POPE, [HK16],…
  - Blind seer, [Lu12], [FJKNRS15],…
  - FH-OPE, Lewi-Wu, Arx, Cipherbase, EncKV,…
- These schemes are surprisingly vulnerable to attack in realistic setting (density + uniform queries + access pattern leakage): **O(*N*log*N*) queries suffice!**
- Even more severe attacks are possible when auxiliary distribution + rank leakage is available.
- Read more at eprint 2017/701.