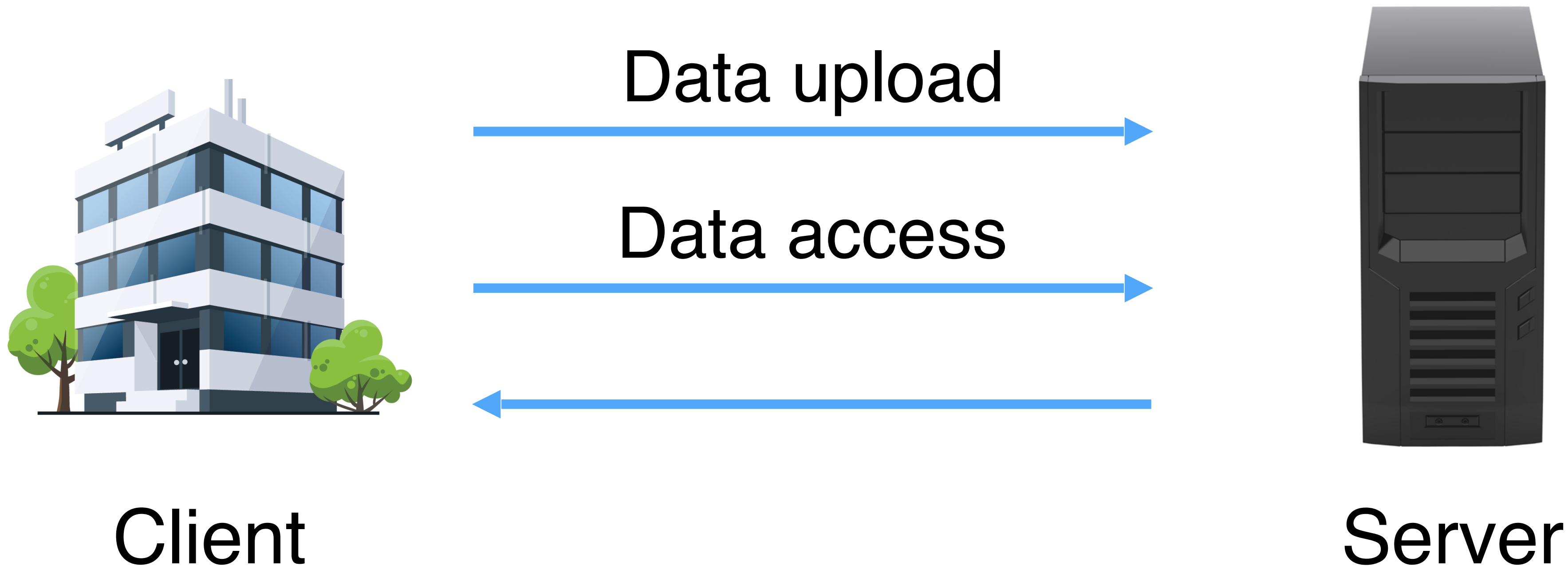
A Review of Database Reconstruction

Brice Minaud (Inria/ENS)

joint work with:
Paul Grubbs (Cornell), Marie-Sarah Lacharité (RHUL), Kenny Paterson (ETH)
[LMP18] (S&P 2018), [GLMP18] (CCS 2018), [GLMP19] (S&P 2019)

ICERM workshop, Brown University, 2019

Outsourcing Data

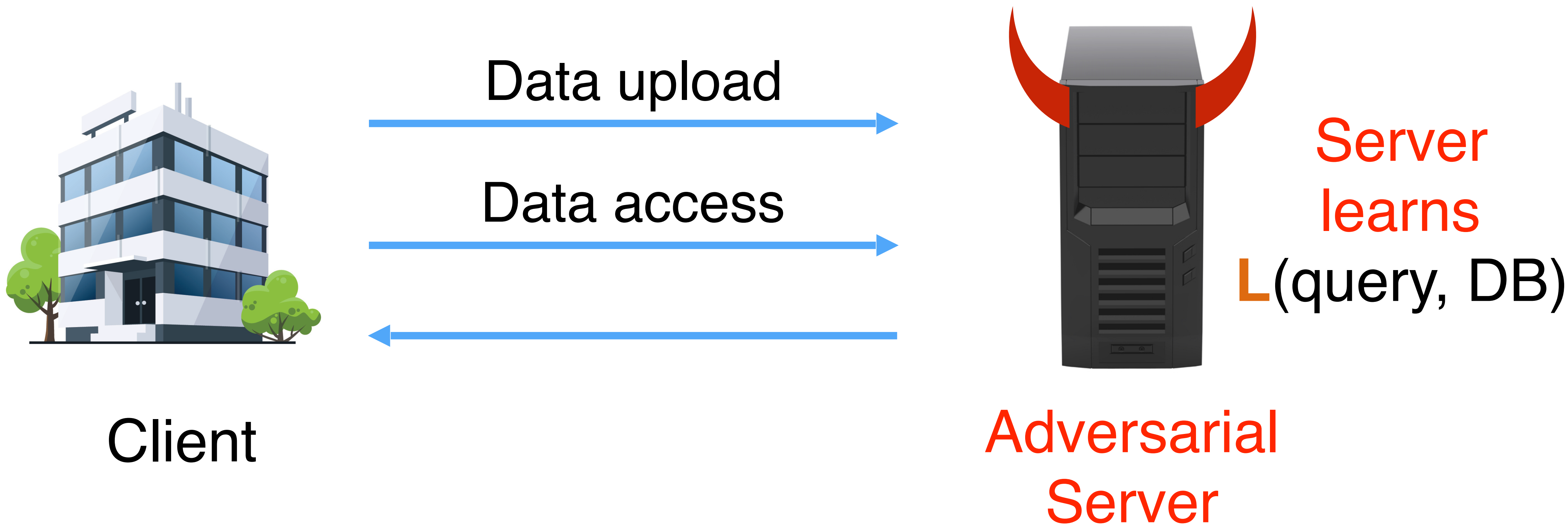


Searchable Encryption: **encrypted database** allowing search queries. In the static case: no updates.

Adversary: **honest-but-curious** host server.

Security goal: **confidentiality** of **data** and **queries**.

Security Model

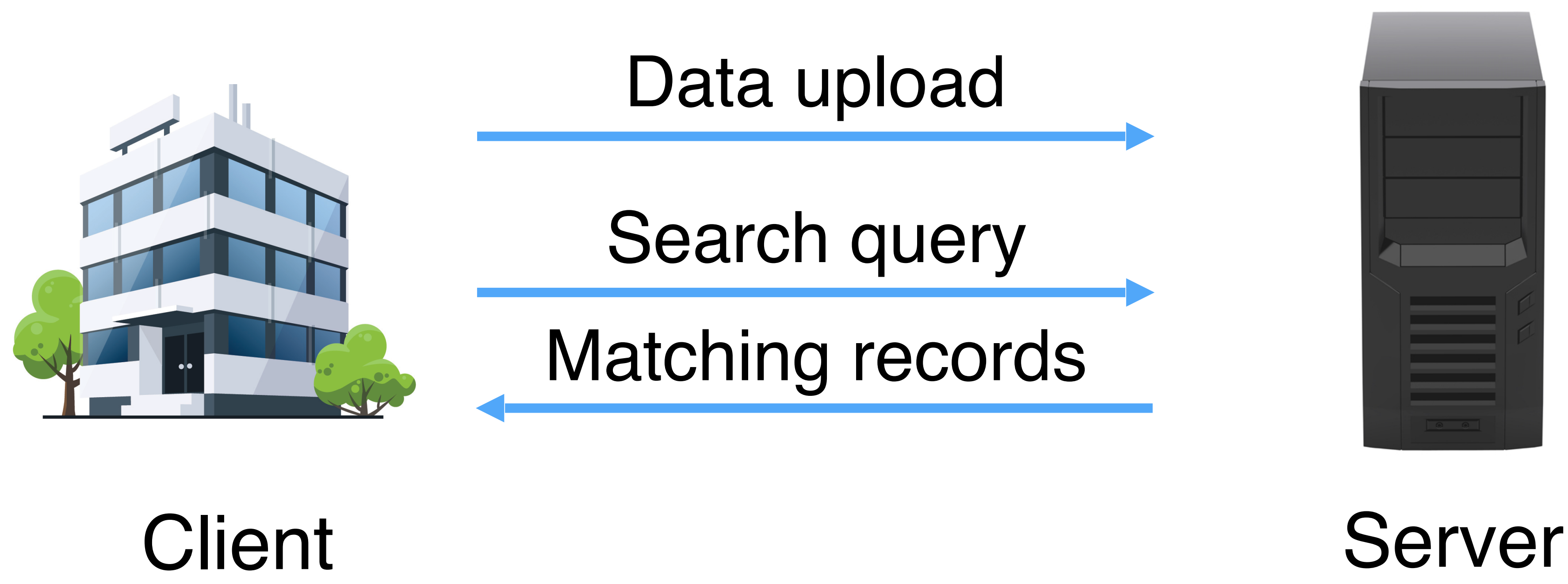


Generic solutions (FHE) are infeasible at scale → for efficiency reasons, some **leakage** is allowed.

Security model: parametrized by a **leakage function L**.

Server learns **nothing** except for the output of the leakage function.

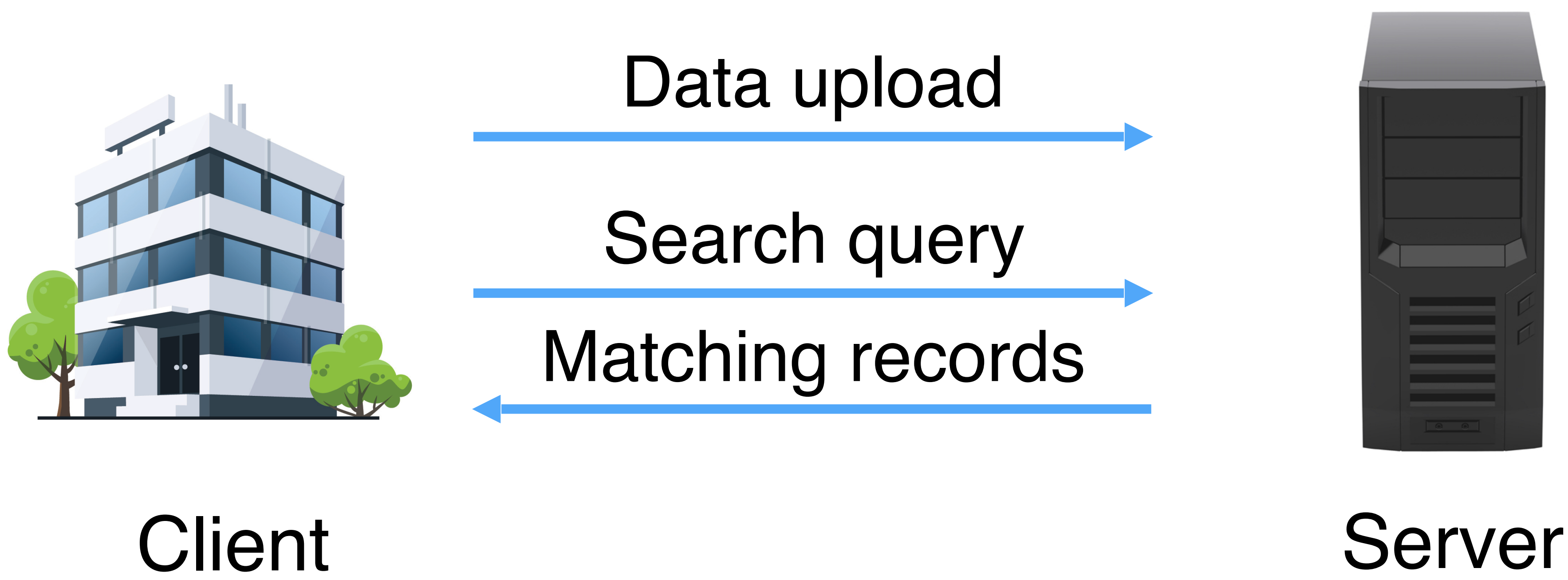
Keyword Search



Symmetric Searchable Encryption (SSE) = keyword search:

- Data = collection of documents. *e.g. messages.*
- Serch query = find documents containing given keyword(s).

Beyond Keyword Search



For an **encrypted database management system**:

- Data = collection of records. *e.g. health records.*
- Basic query examples:
  - find records with given value. *e.g. patients aged 57.*
  - find records within a given range. *e.g. patients aged 55-65.*

Range Queries

In this talk: **range queries**.

- Fundamental for any encrypted DB system.
- Many constructions out there.
- Simplest type of query that can't “just” be handled by an index.

Natural solutions:

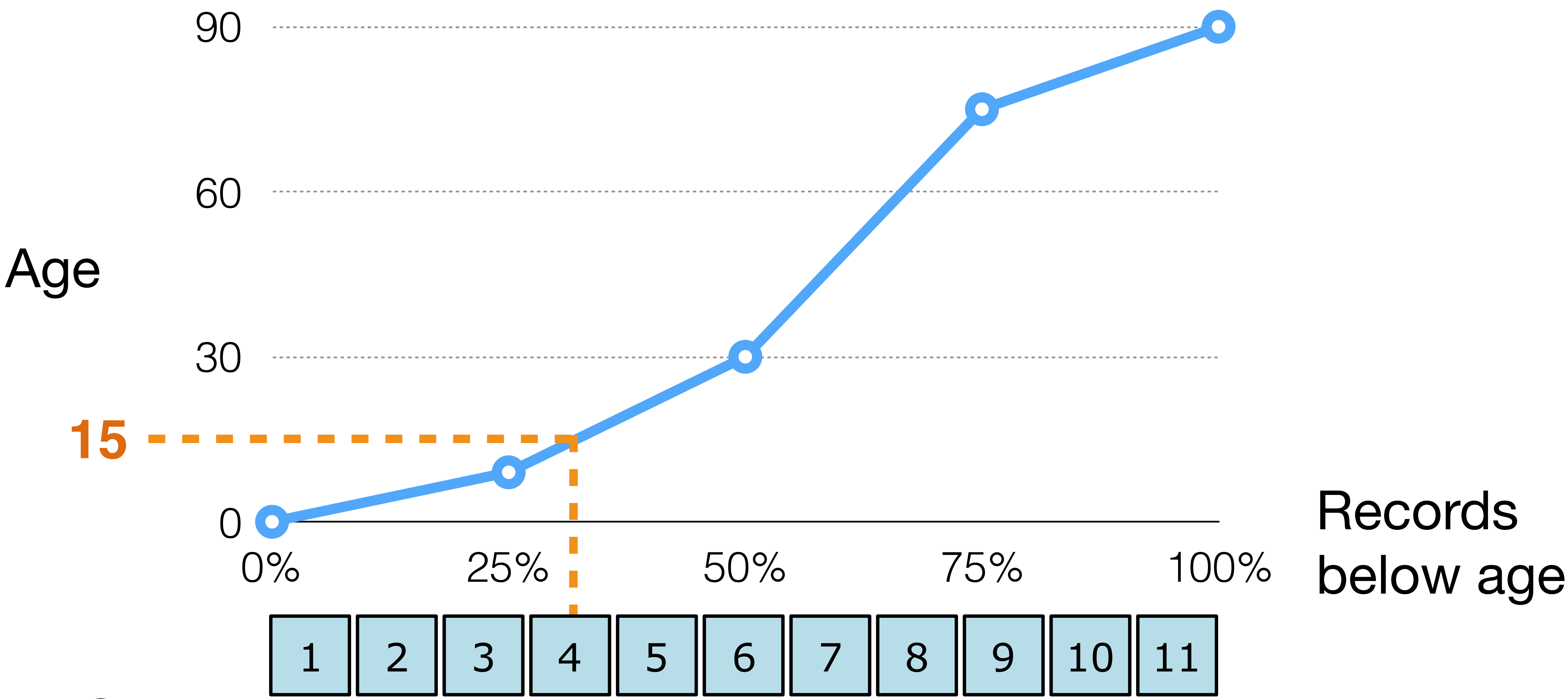
Order-Preserving, **Order-Revealing Encryption**.

- Plaintexts are **ordered**, ciphertexts are **ordered**.
- The encryption map **preserves order**.

Attacks Exploiting ORE*

- **"Sorting" attack**: if every possible value appears in the DB... Just sort the ciphertexts and you learn their value!

- **"CDF-matching" attack**: say the attacker has an approximation of the **Cumulative Distribution Function** of DB values...



*not L/R ORE.

Leakage-Abuse Attacks

“Leakage-abuse attacks” (coined by Cash et al. CCS'15):

- Do not contradict security proofs.
- Can be devastating in practice.

ORE: order information can be used to infer (approximate) values.
Leaking order is too revealing.

→ **“Second-generation” schemes** enable range queries *without* relying on OPE/ORE.

Cryptanalysis and Leakage Abuse

What is the point of these attacks?

- Understand concrete security implications of leakage.
- "Impossibility results" → help guide design.

Approach: consider *general* settings. Pioneered by [KKNO16].

Here:

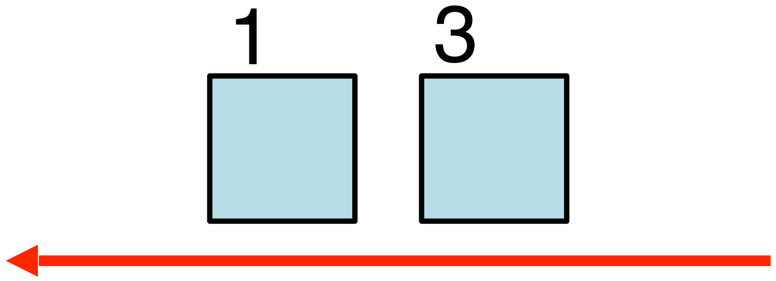
- Range queries.
- Passive, persistent adversary. No injections, no chosen queries.

Roadmap

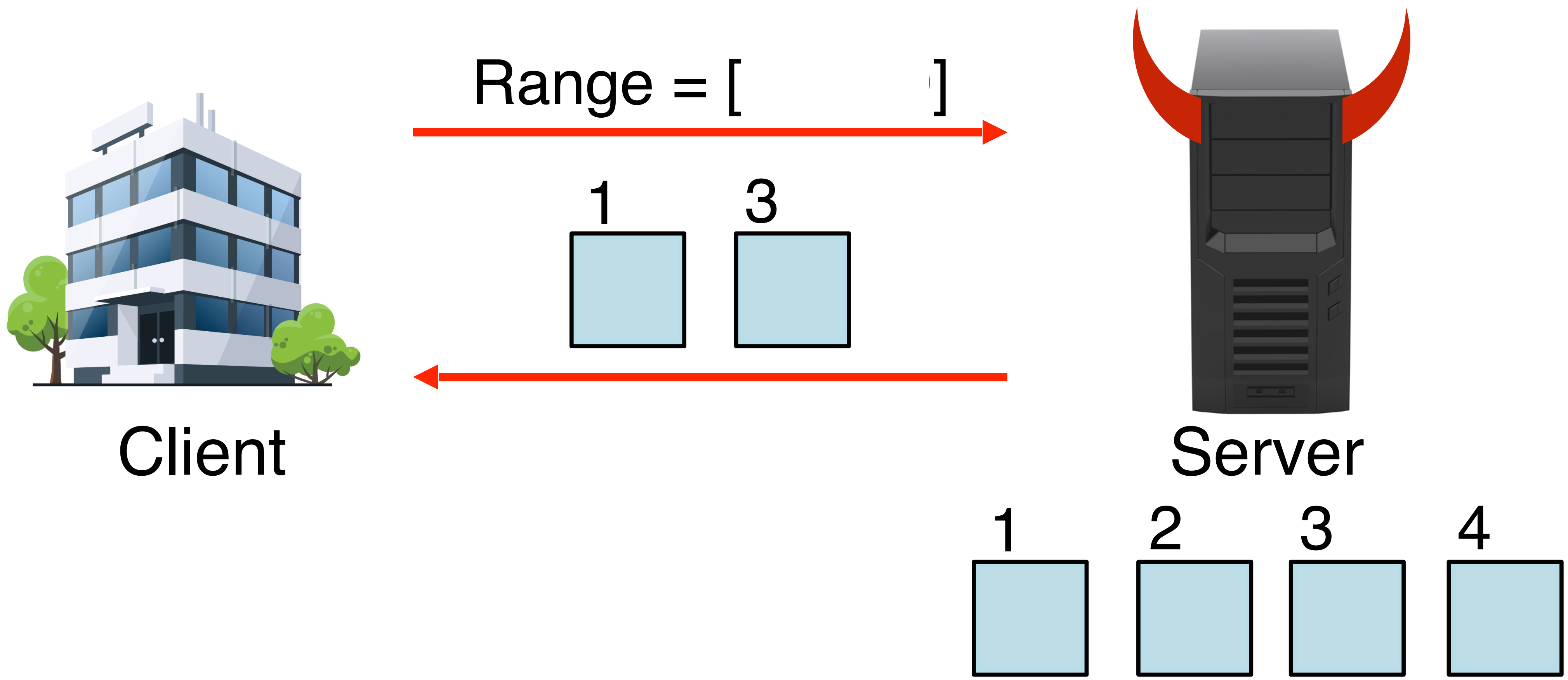
1. Access pattern leakage.

3. Volume leakage.

Access Pattern Leakage



Range Queries

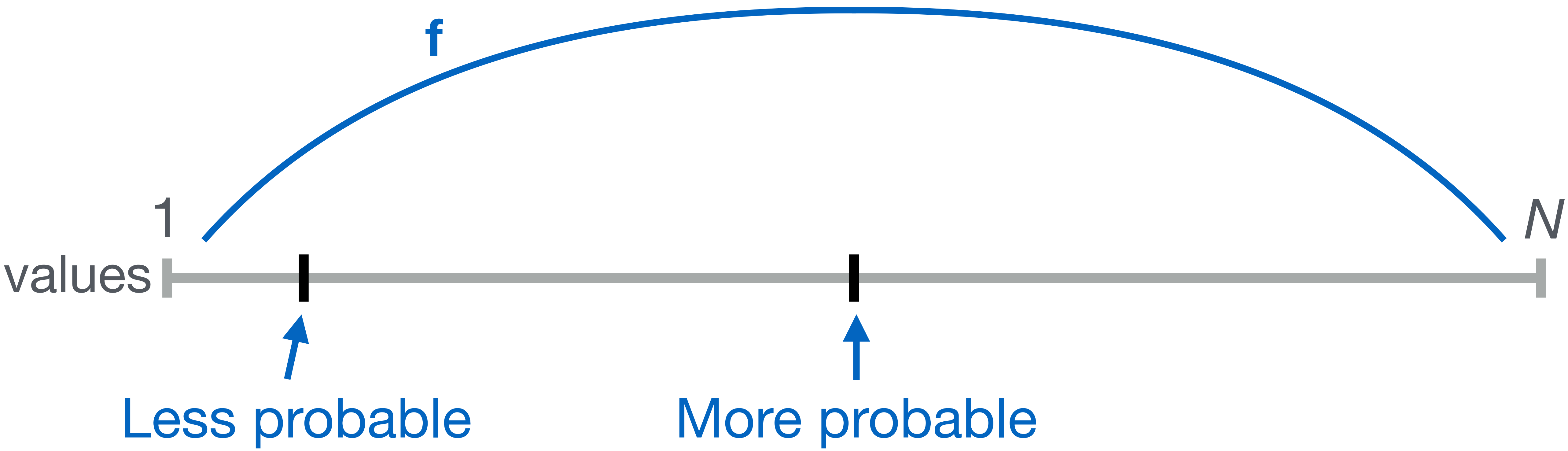


SE schemes supporting range queries are proven secure w.r.t. a leakage function including **access pattern leakage**.

What can the server learn from the above leakage?

Let $\boldsymbol{N}$ = number of possible values.

KKNO16 Attack



Assume a **uniform distribution** on range queries.

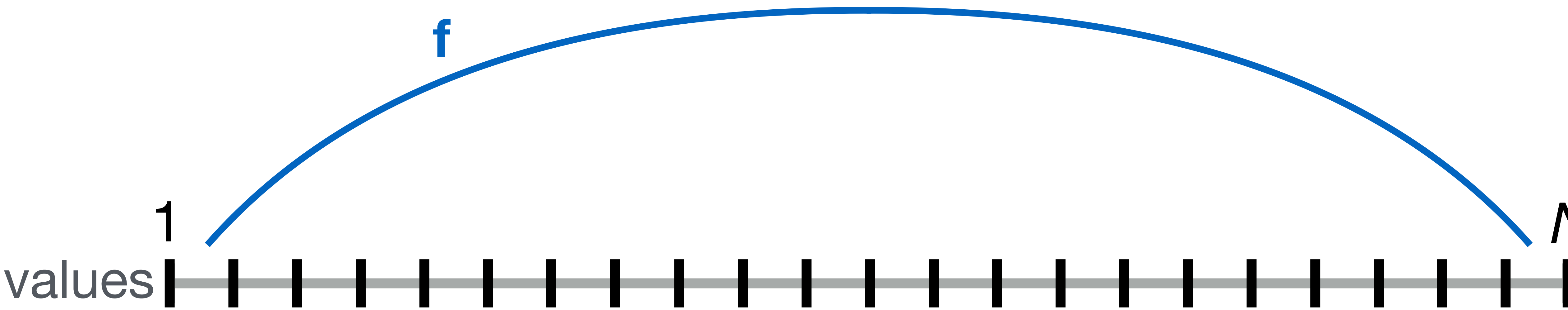
Induces a distribution **f** on the prob. that a given value is hit.

Idea: for each record...

1. Count frequency at which the record is hit.
→ gives estimate of probability it's hit by uniform query.

2. deduce estimate of its value by "inverting" **f**.

KKNO16 Attack



Step 1: for **every** record, estimate prob of the record being hit.

Step 2: "invert" **f**.

Step 3: break the symmetry, i.e. reconcile which values are on the same side of N/2.

After $O(N^4 \log N)$ uniform queries, previous alg. recovers the *exact* value of *all* records.

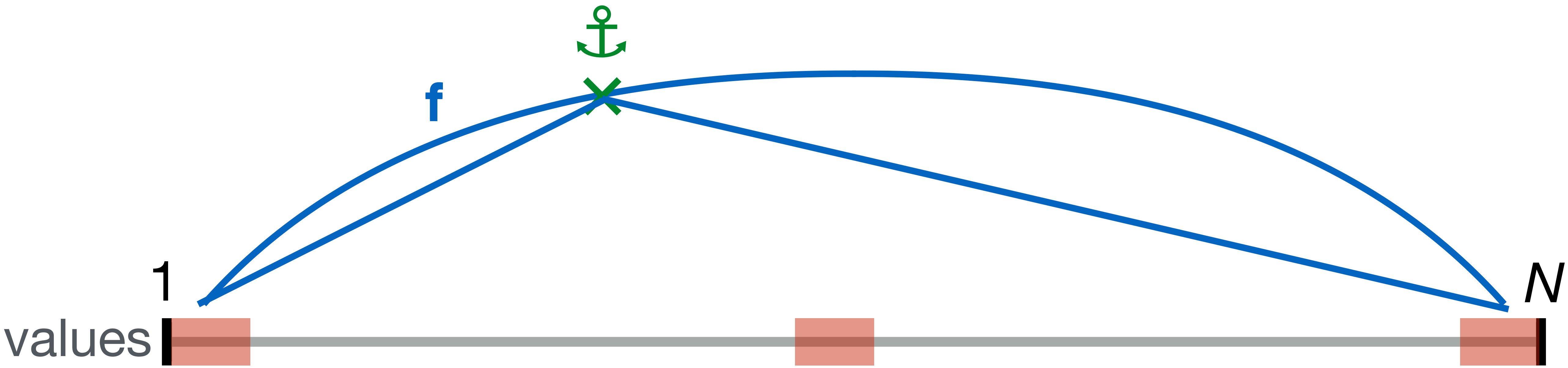
KKNO16 Attack

After $O(N^4 \log N)$ uniform queries, previous alg. recovers the *exact* value of *all* records.

Remarks:

- Requires **uniform** distribution.
- **Expensive**. In fact, uses up *all possible* leakage information!
- Lower bound of $\Omega(N^4)$.

Revisiting the Analysis, Part I [GLMP19]



Step 0: find suitable "anchor" record.

Step 1: for **every** record, estimate distance to anchor.

Step 2: "invert" **f**. ← costs a **constant** factor!

Step 3: break the symmetry, i.e. reconcile which values are on the same side of N/2.

After $O(N^2 \log N)$ uniform queries, previous alg. recovers the *exact* value of *all* records.

Cheaper KKNO16 attack

After $O(N^2 \log N)$ uniform queries, previous alg. recovers the *exact* value of *all* records.

Remarks:

- Requires **uniform** distribution.
- Requires existence of a favorably placed record.
- **Still fairly expensive**.
- Lower bound of $\Omega(N^2)$. Can't hope to get below.

Approximate Reconstruction

Strongest goal: **full** database reconstruction = recovering the exact value of every record.

More general: **approximate** database reconstruction = recovering all values within εN.

$\varepsilon = 0.05$ is recovery within 5%. $\varepsilon = 1/N$ is full recovery.

(“Sacrificial” recovery: values very close to 1 and N are excluded.)

Database Reconstruction

[KKNO16]: full reconstruction in $O(N^4 \log N)$ queries.

recovers

[GLMP19]:

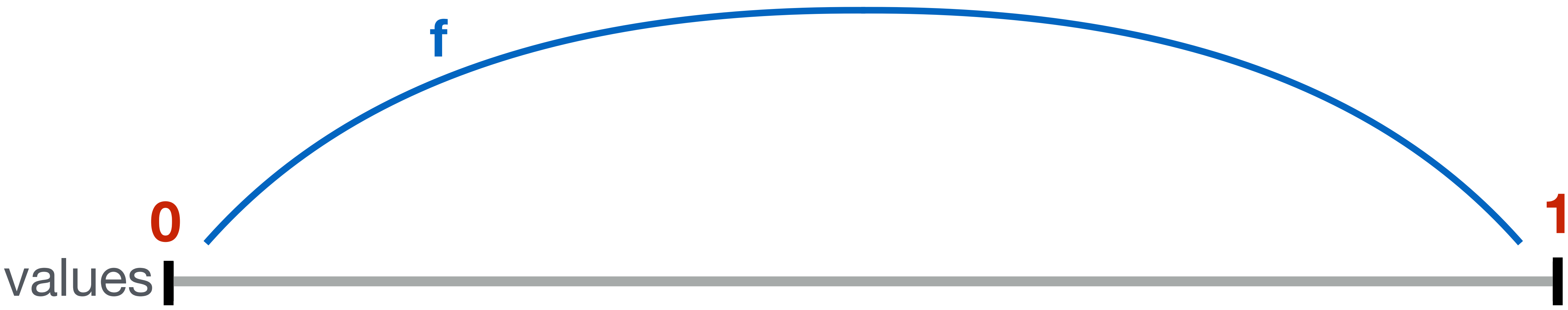
	Full. Rec.	Lower Bound
‣ $O(\varepsilon^{-4} \log \varepsilon^{-1})$ for approx. reconstruction.	$\boxed{O(N^4 \log N)}$	$\Omega(\varepsilon^{-4})$
‣ $O(\varepsilon^{-2} \log \varepsilon^{-1})$ with mild hypothesis.	$O(N^2 \log N)$	$\Omega(\varepsilon^{-2})$

Scale-free: does not depend on size of DB or number of possible values.

→ Recovering all values in DB within 5% costs O(1) queries!

Analysis: uses VC theory + draws connection to machine learning. See Paul's talk!

Intuition for Scale-Freeness



Step 1: for **every** record, estimate prob of the record being hit.

Step 2: “invert” **f**.

Instead of support = integers 1 to N, take reals [0,1].

...so “$N = \infty$” !

The previous algorithm still works!

On the i.i.d. Assumption

+ Scale-freeness. *N* and DB size irrelevant for query complexity.

- We are assuming **uniformly distributed** queries.

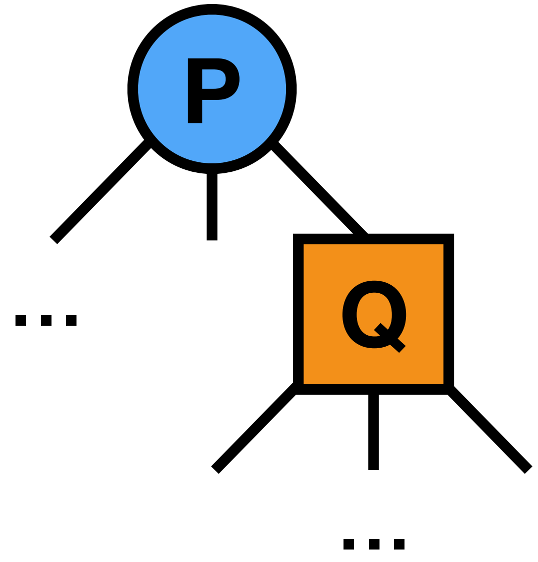
In reality we are assuming:

- Queries are **uniform**.
- The **adversary knows** the query distribution.
- Queries are **independent and identically distributed**.

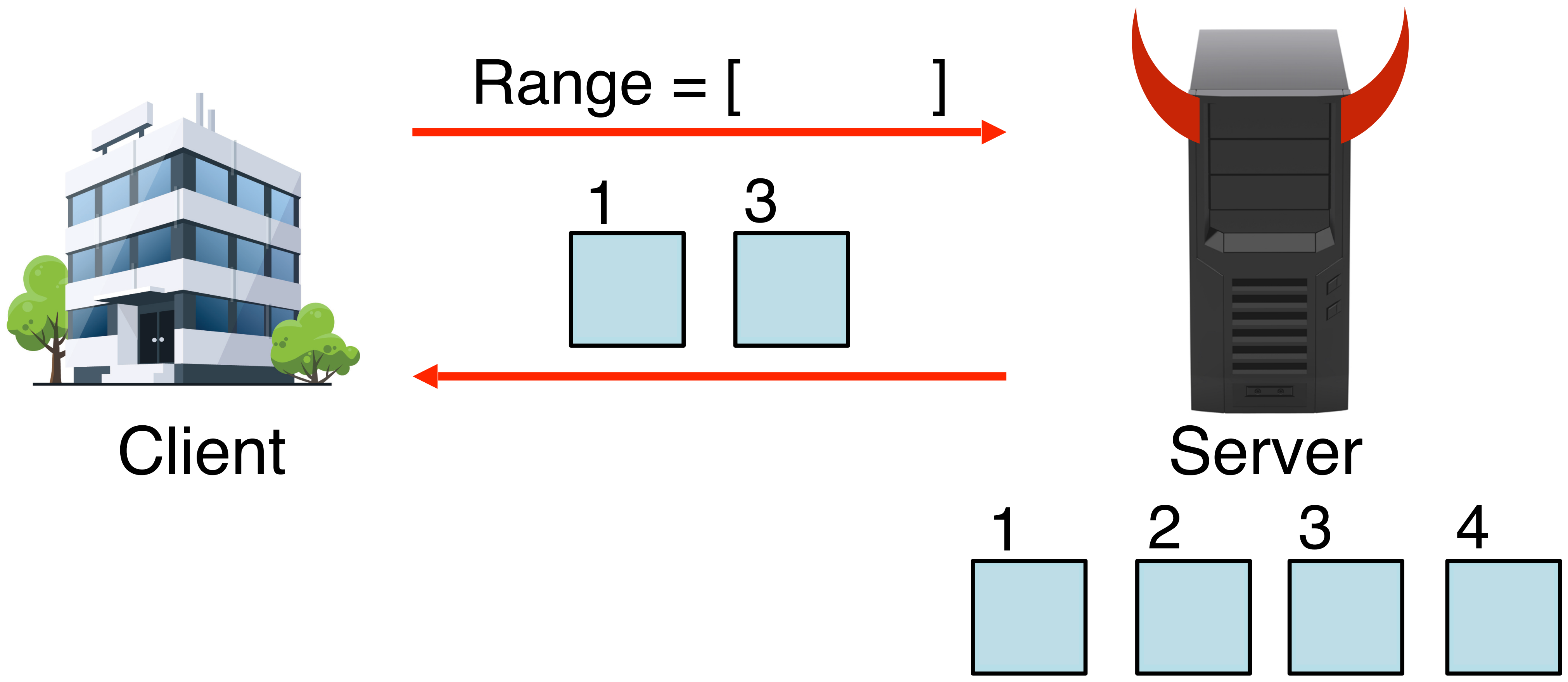
This is not realistic.

What can we learn without that hypothesis?

Order Reconstruction



Problem Statement



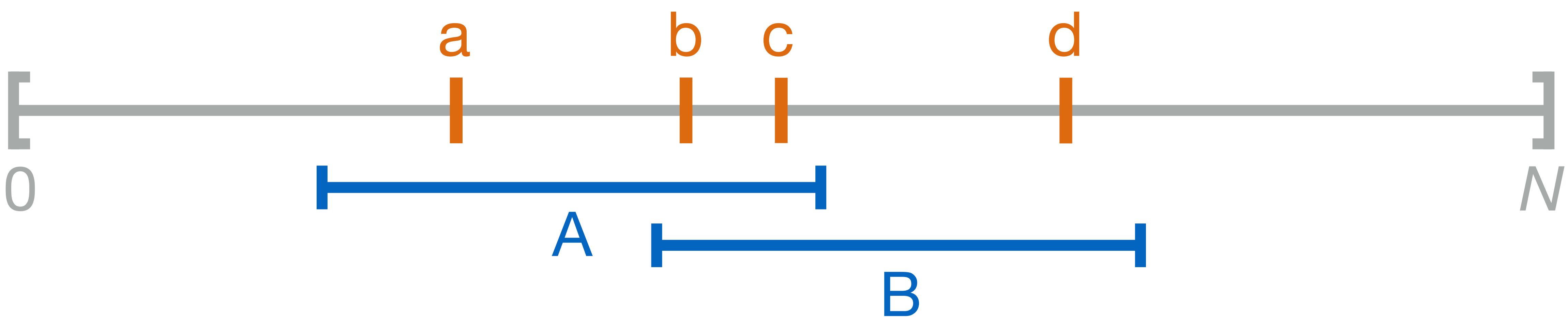
What can the server learn from the above leakage?

This time we **don't assume** i.i.d. queries, or knowledge of their distribution.

Range Query Leakage

Query A matches records a, b, c.

Query B matches records b, c, d.



Then this is the only configuration (up to symmetry)!

→ we learn that records b, c are *between* a and d.

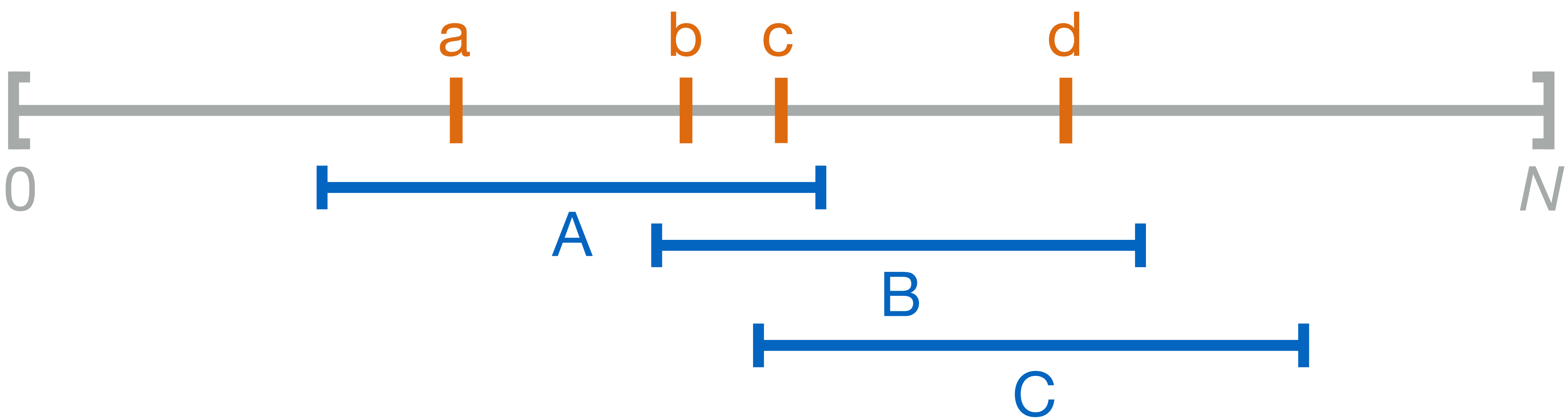
We learn something about the **order** of records.

Range Query Leakage

Query A matches records a, b, c.

Query B matches records b, c, d.

Query C matches records c, d.



Then the only possible order is a, b, c, d (or d, c, b, a)!

Challenges:

- How do we extract order information? (What **algorithm**?)
- How do we **quantify** and **analyze** how fast order is learned as more queries are observed?

Challenge 1: the Algorithm

Short answer: there is already an algorithm!

Long answer: **PQ-trees**.

X: linearly ordered set. Order is unknown.

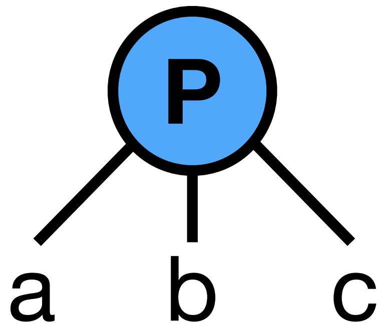
You are given a set S containing some intervals in X.

A **PQ tree** is a compact (linear in $|X|$) representation of the set of all permutations of X that are compatible with S.

Can be updated in linear time.

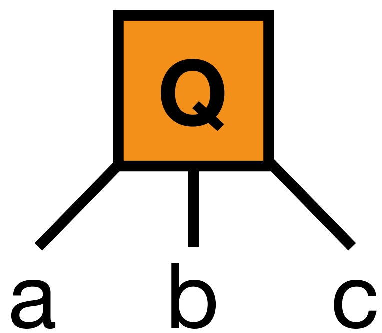
Note: was used in [DR13], didn't target reconstruction.

PQ Trees



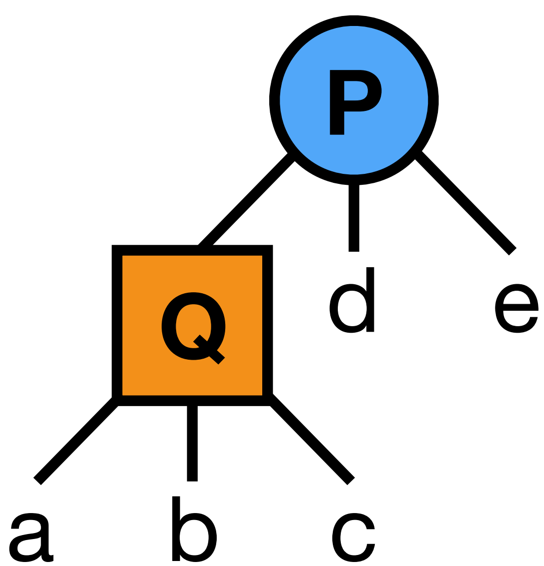
Order is completely **unknown**.

- any permutation of abc.



Order is completely **known** (up to reflection).

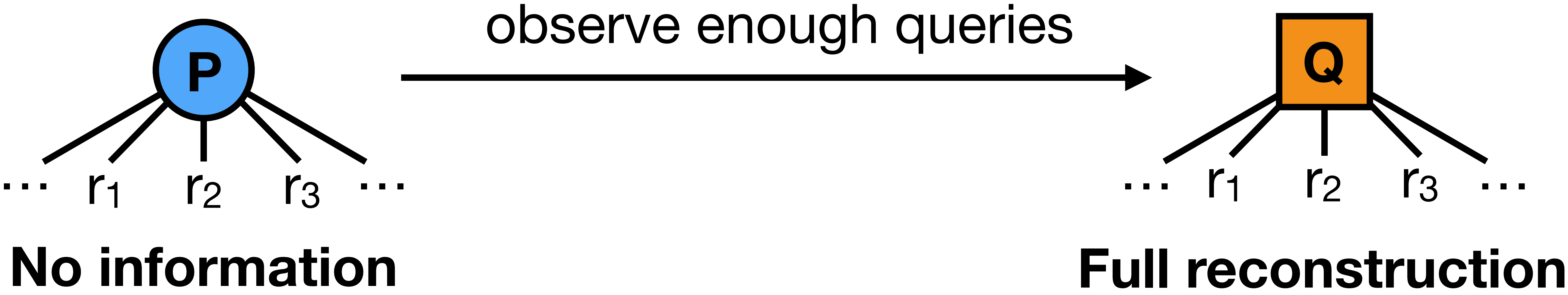
- abc’or ‘cba’.



Combines in the natural way.

- ‘abcde’, ‘abced’, ‘dabce’, ‘eabcd’, ‘deabc’, ‘edabc’, ‘cbade’ etc.

Full Order Reconstruction



We want to **quantify** order learning...

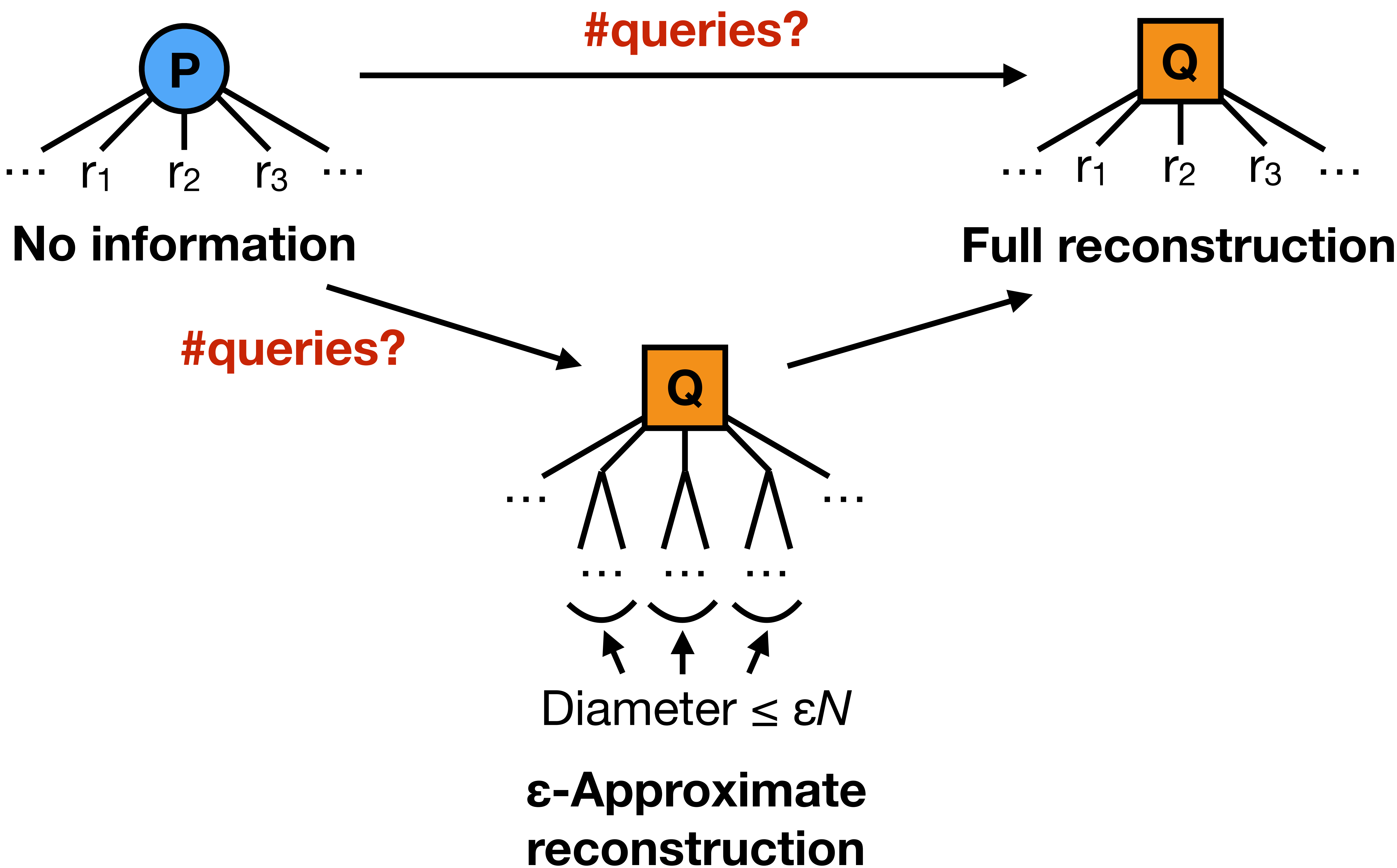
Challenge 2a: Quantify Order Learning

ε-Approximate order reconstruction.

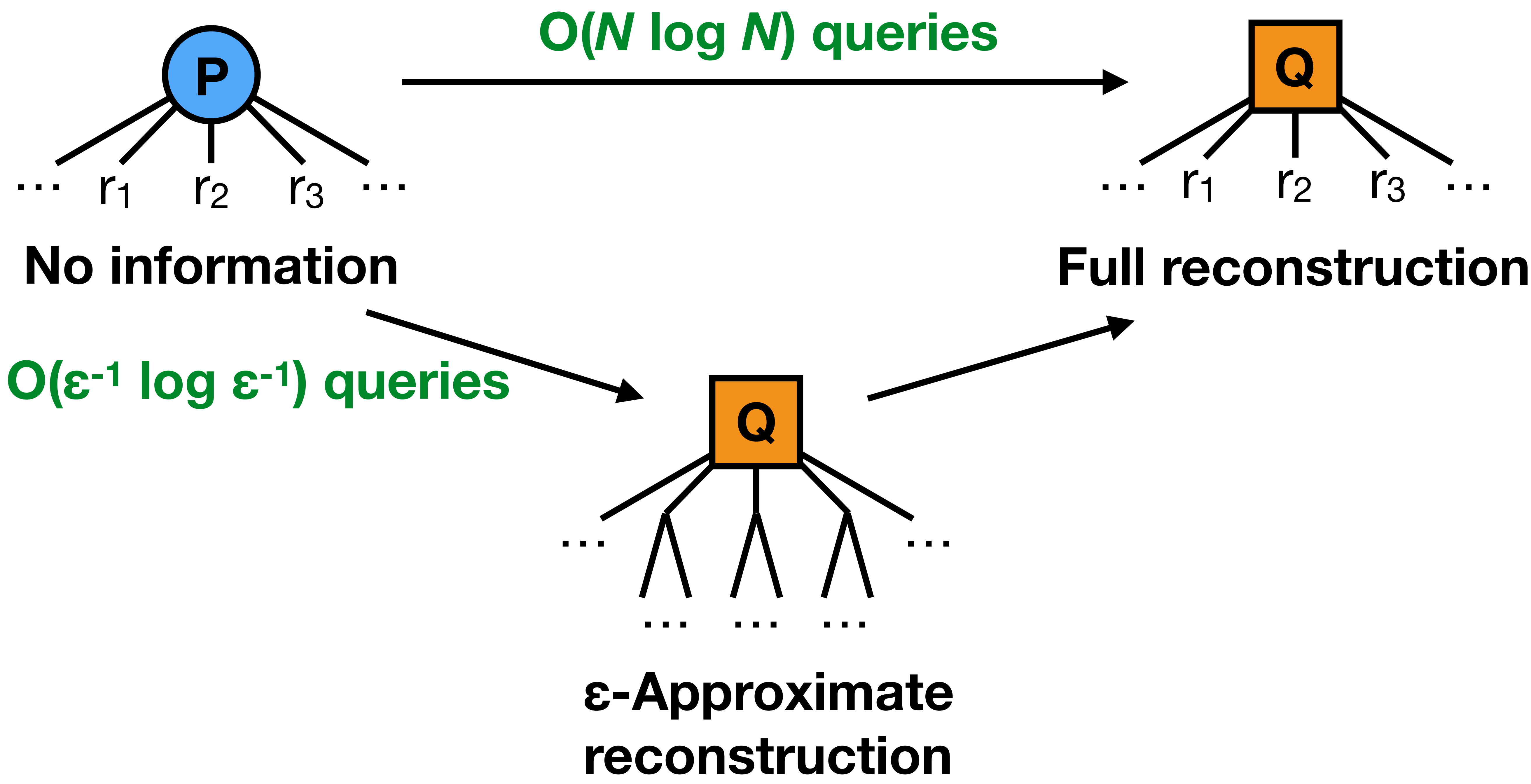
Roughly: we learn the order between two records as soon as their values are $\geq \varepsilon N$ apart. ($\varepsilon = 1/N$ is full reconstruction)

Note: compatible with "ORE-style" CDF matching.

Approximate Order Reconstruction



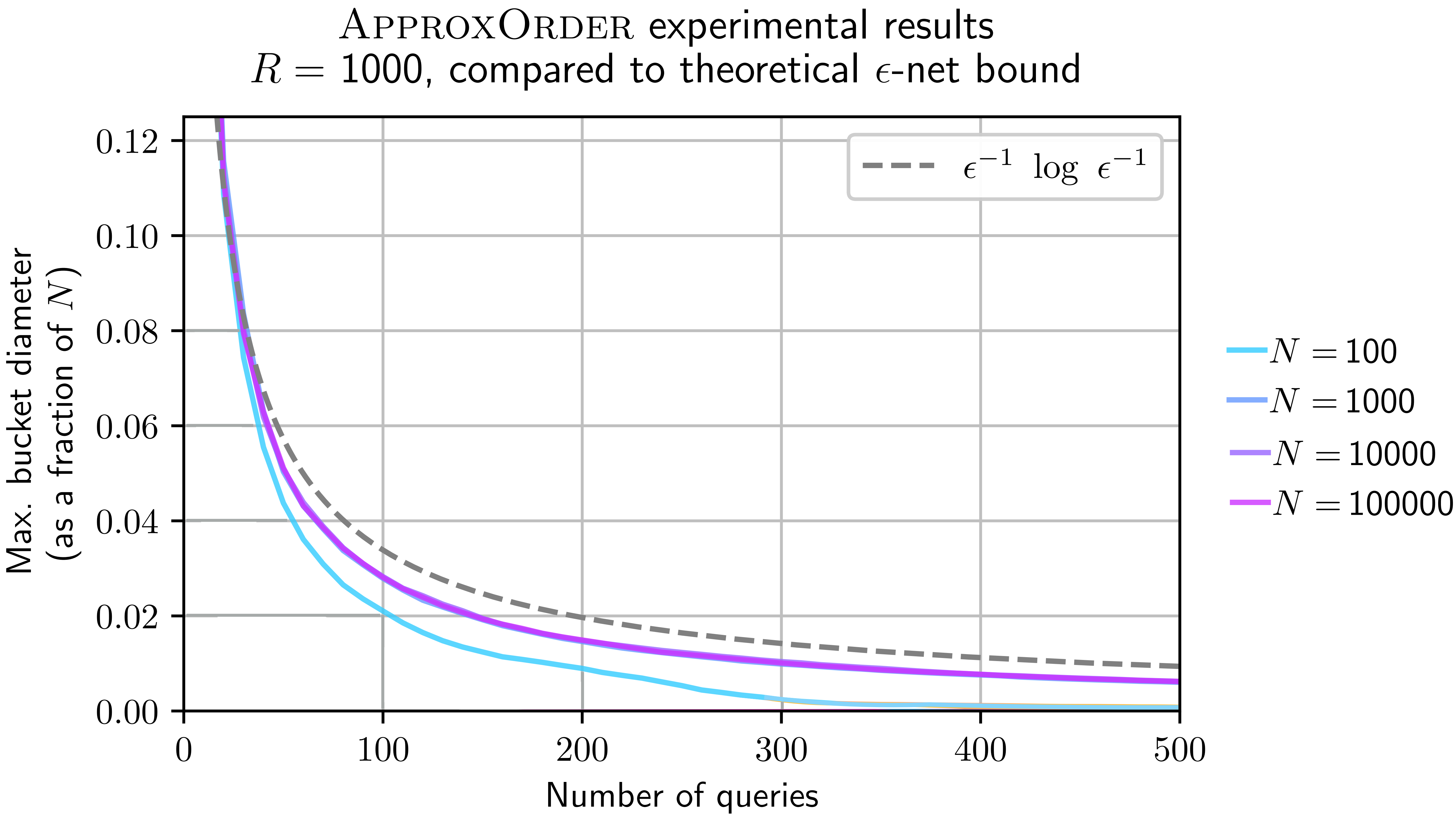
Approximate Order Reconstruction



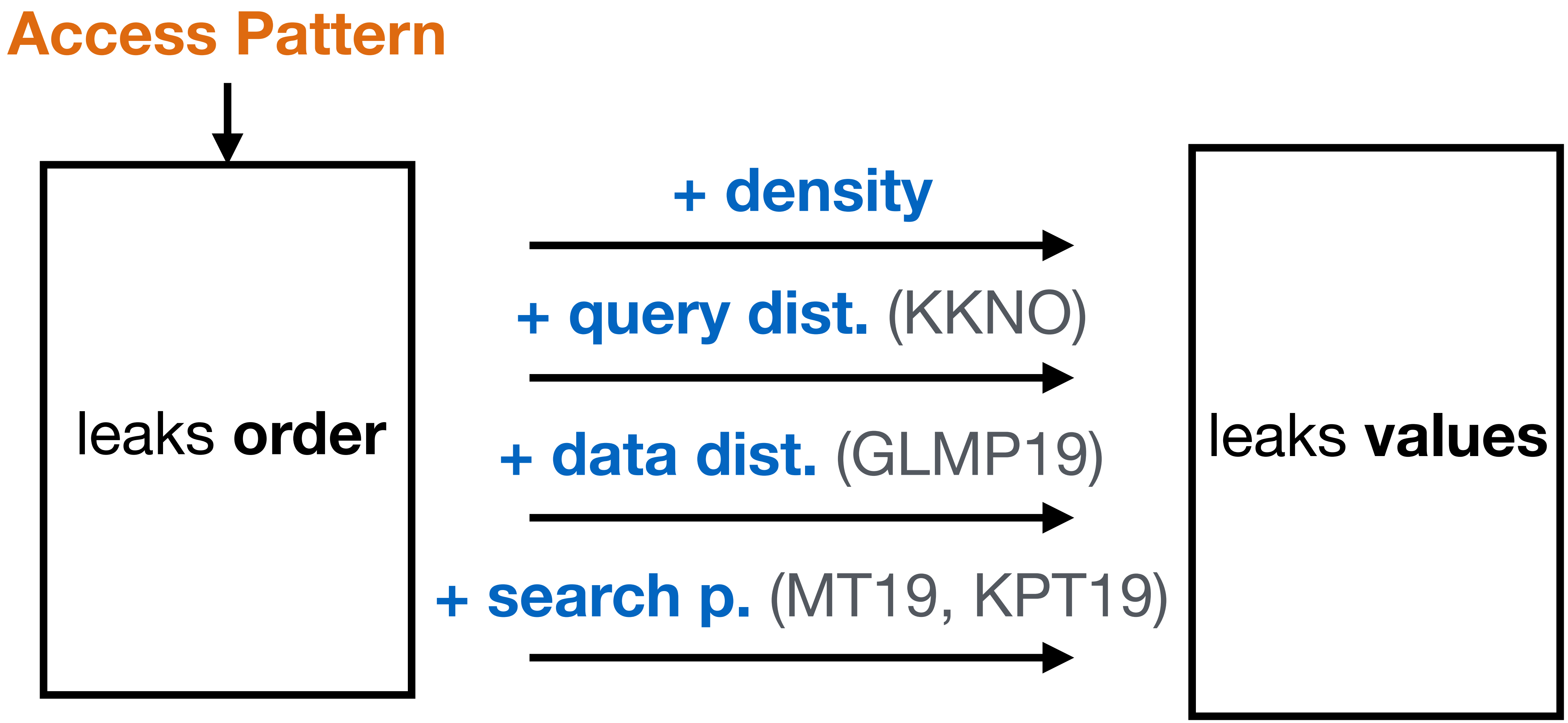
Conclusion: learn order very quickly.

Note: some (weak) assumptions are swept under the rug.

Experiments

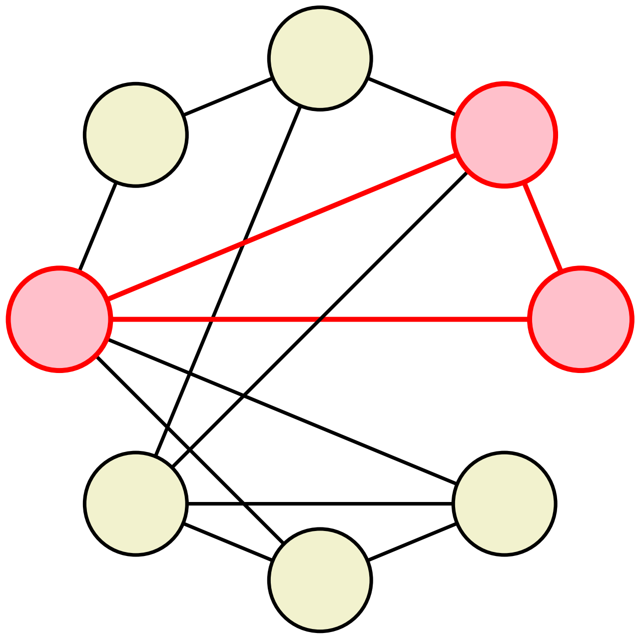


Big Picture

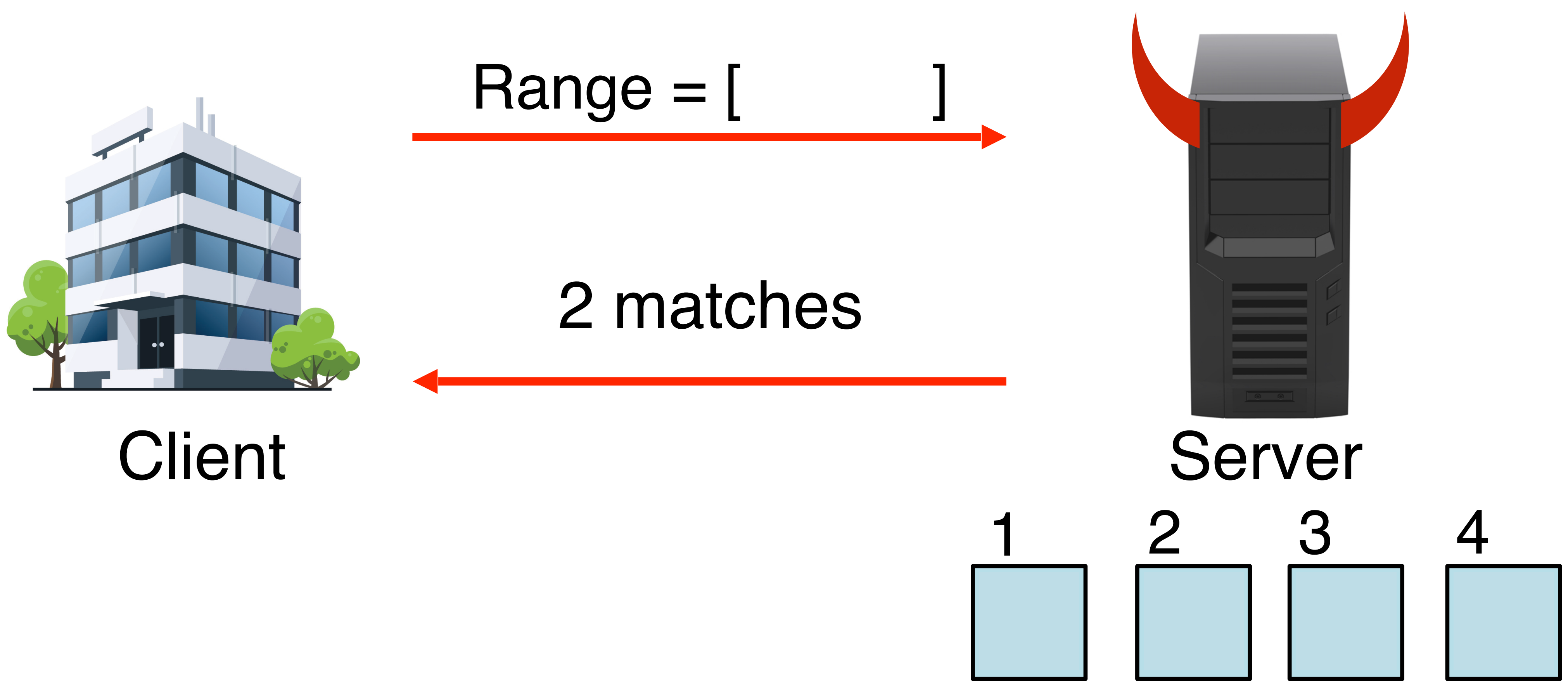


- **Resilient**, scale-free attacks.
- Effective in practice in some realistic scenarios.
- Watch out for additional leakage. E.g.:
  - Search pattern.
  - Rank information (e.g. L/R ORE). Damaging for low #queries.

Volume Leakage



Problem Statement

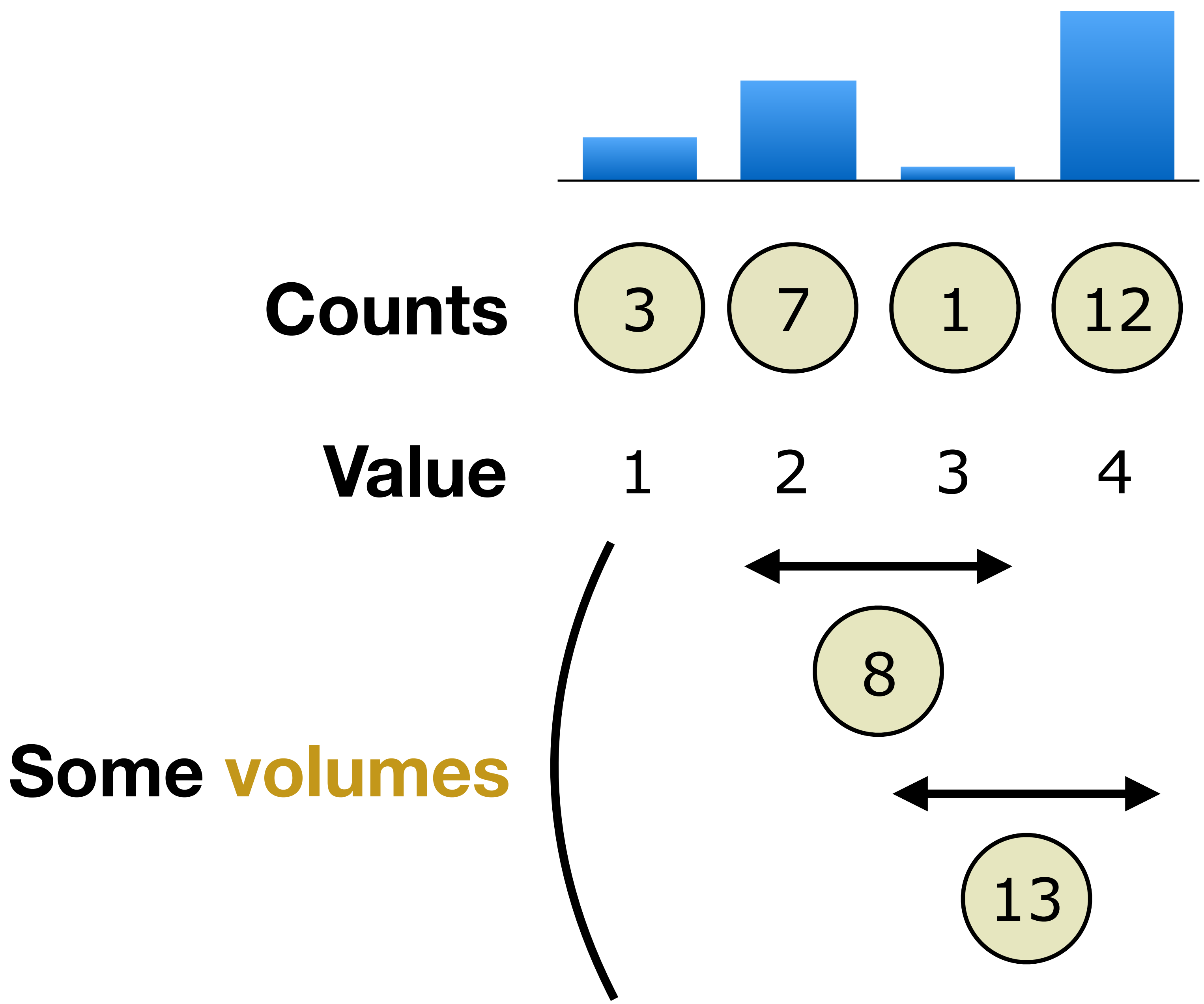


Attacker *only* sees **volumes** = **number of records** matching each query.

What can the server learn from the above leakage?

Volumes

The attacker wants to learn exact **counts.**



A **volume** = number of records matching some range.

KKNO16 Volume Attack

Assume **uniform** queries.

Step 1: recover exact probability of every volume ➔ number of queries that have each volume.

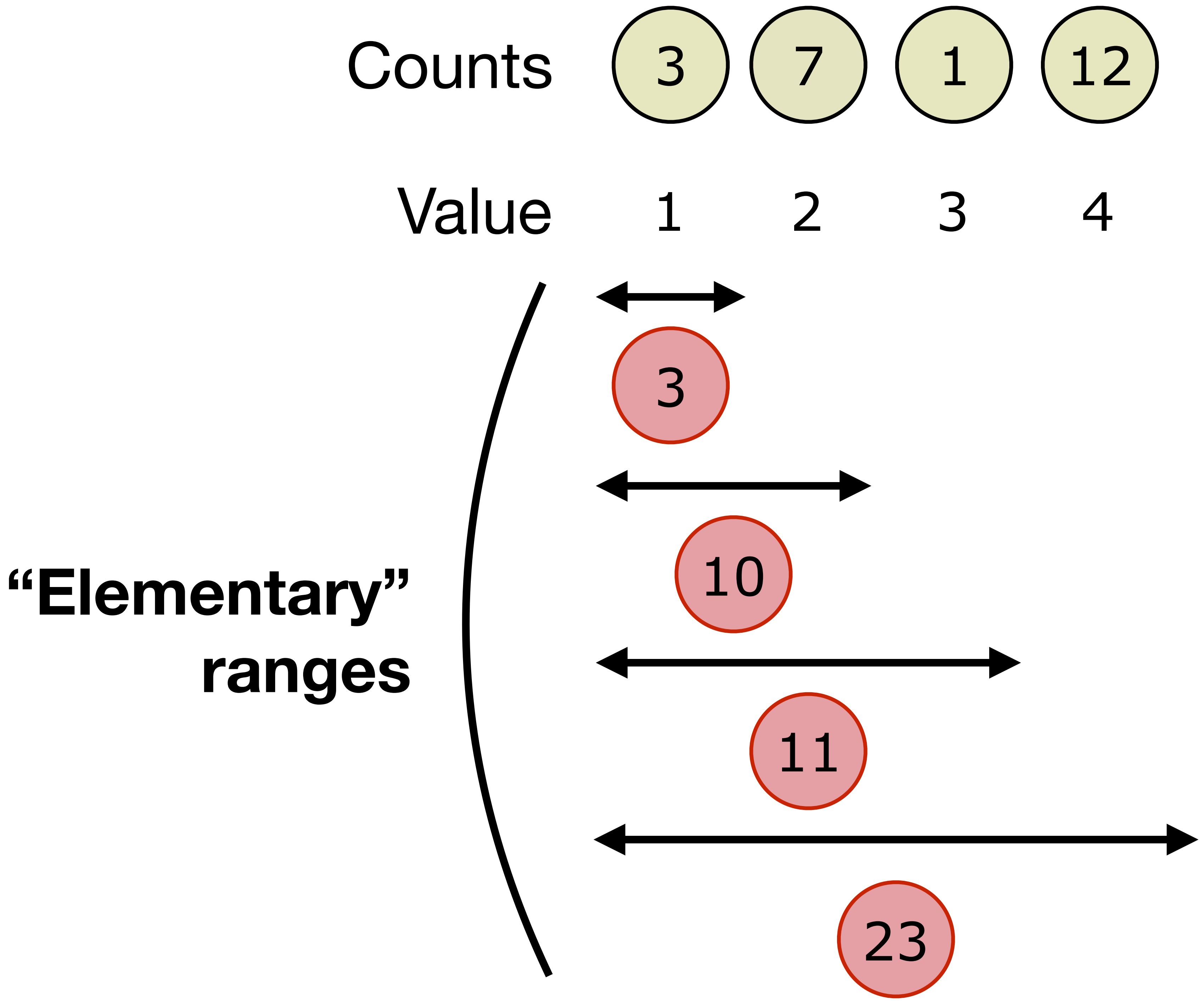
Step 2: express and solve equation system linking above data back to DB counts. (Ends up as polynomial factorization.)

After $O(N^4 \log N)$ uniform queries, previous alg. recovers all DB counts.

Remarks:

- Requires **uniform** distribution.
- **Expensive**. In fact, uses up *all possible* leakage information!
- Lower bound of $\Omega(N^4)$.

Elementary Volumes [GLMP18]



Elementary volumes = volumes of ranges [1,1], [1,2], [1,3]...

Elementary Volumes

Counts	3	7	1	12
Value	1	2	3	4

Fact:

$$\boxed{\mathrm{vol}([a,b]) = \mathrm{vol}([1,b]) - \mathrm{vol}([1,a])}$$

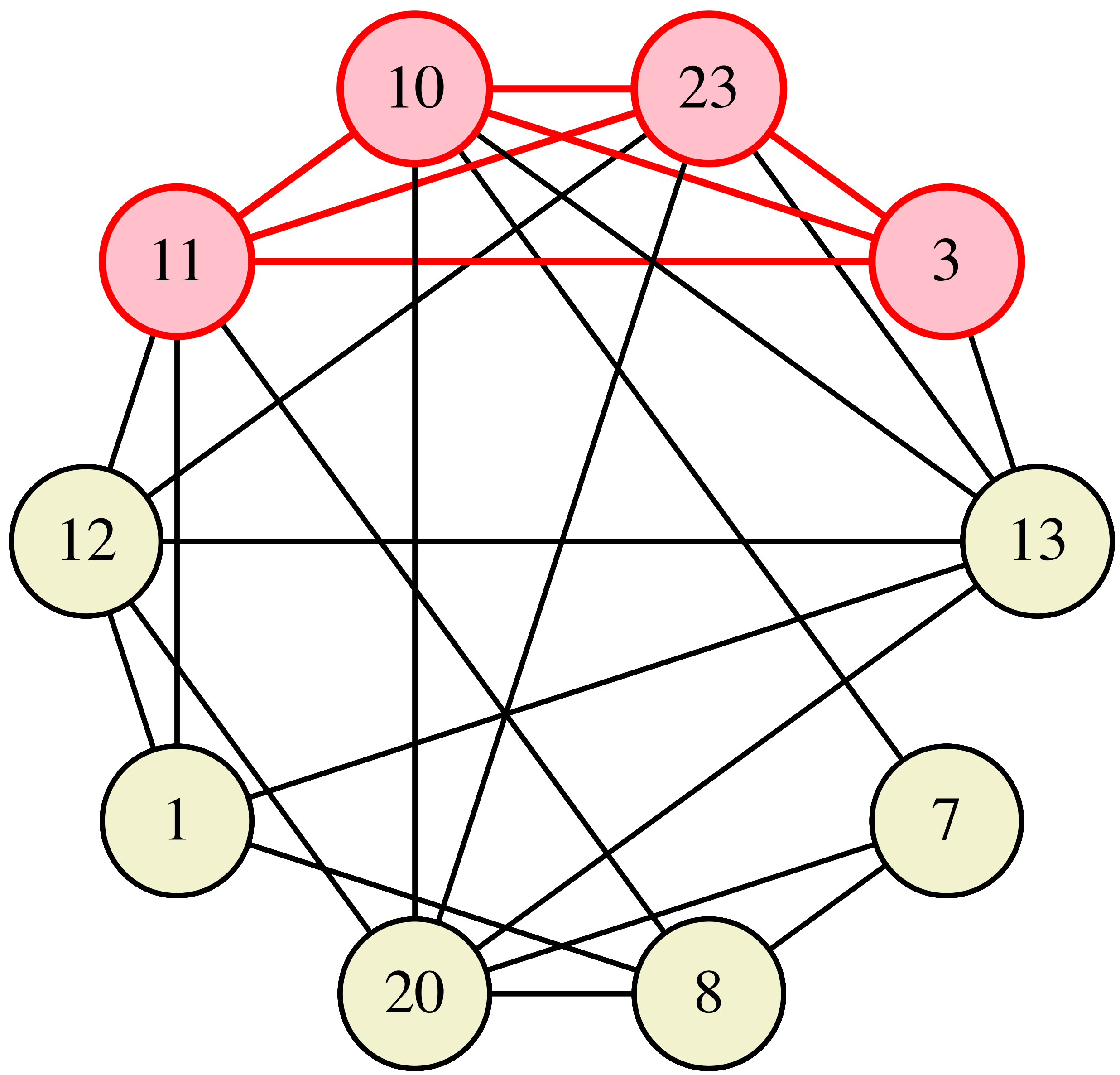
so...

- Every volume is = difference of two elementary volumes.
- Knowing set of elementary volumes ⇔ knowing counts.

Our goal: finding elementary volumes.

The Attack

Assumption: the volumes of all queries are observed.



Draw an **edge** between volumes **a** and **b** iff **|b-a|** is a volume.

Summary

Attack: **elementary volumes** form a clique in the volume graph → clique-finding algorithm reveals them.

For structured queries, even just volume leakage can be quite damaging. Attack requires strong assumption.

In the article:

- *Pre-processing to avoid clique finding.*
- *Analysis of parameters + experiments.*
- *Other attacks.*

Conclusion

Conclusion

Access pattern:

- **Resilient**, scale-free attacks.
- Effective in practice in some realistic scenarios.

➔ non-trivial countermeasures are required.

Volume attacks:

- **Fragile attacks**. Currently.
- Expensive query complexity $O(N^2 \log N)$.
- Unsatisfactory: limits of attacks not clear.

➔ “simple” countermeasures might be enough in most scenarios.

Some open problems: mixed queries, scale-free volumes.