SIGMOD 2024

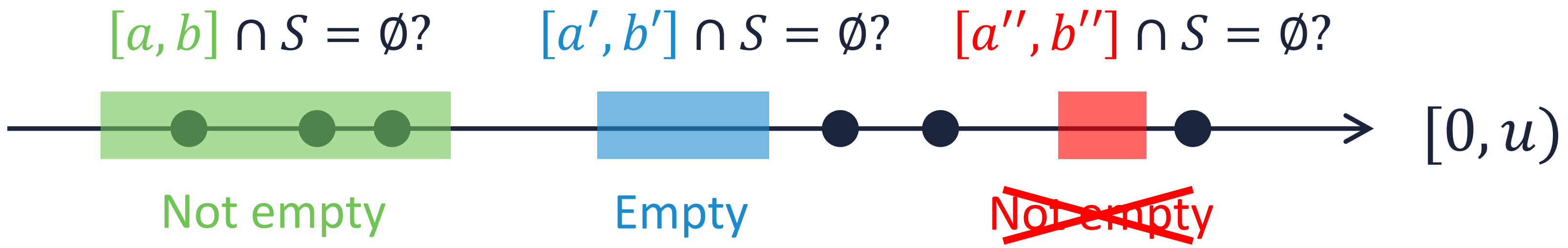
Grafite: Taming Adversarial Queries with Optimal Range Filters

Marco Costa, Paolo Ferragina, and Giorgio Vinciguerra

Università di Pisa

Range filters

Given a set S of n keys, a *range filter* is a space-efficient data structure that answers range emptiness queries



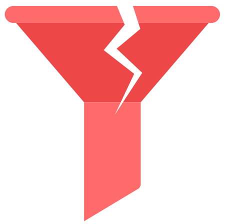
…with a false positive probability of at most ε

- Generalise Bloom filters from point to range queries
- Reduce I/Os of range queries in LSM-based storage engines

State-of-the-art range filters

ARF [VLDB ’13], **SuRF** [SIGMOD ’18], **Rosetta** [SIGMOD ’20], **Proteus** [SIGMOD ’22], **SNARF** [VLDB ’22], **bloomRF** [EDBT ’23], **REncoder** [ICDE ’23], **Oasis** [VLDB ’24], **GRF** [SIGMOD ’24]

Highly complex
Sophisticated designs,
hard to evaluate and deploy

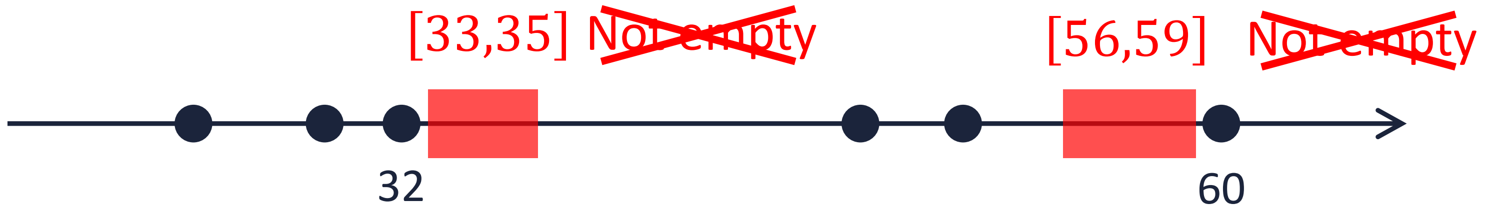


Fragile
Inconsistent FPR and query times across
different datasets and workloads

Goswami et al. [SODA ’15] gave theoretically optimal upper and lower bounds

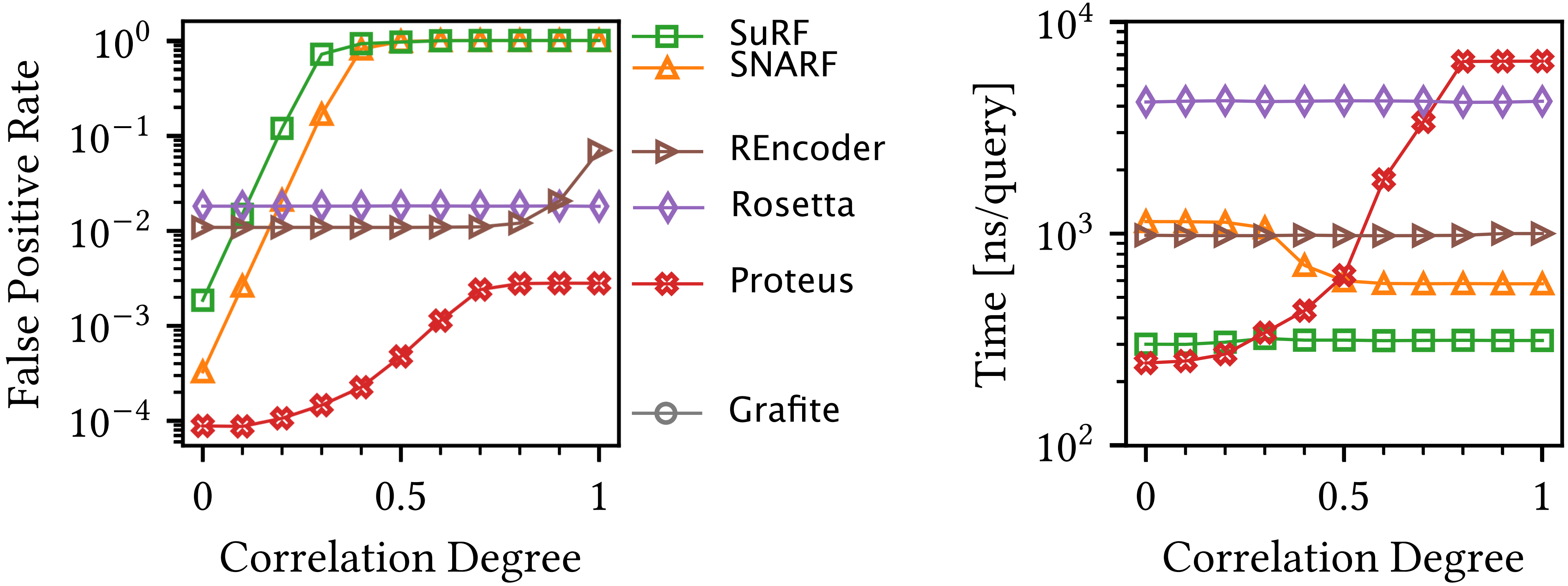
Correlated queries

Most range filters suffer from high false positive rates
when query ranges are close to the input keys



The most expected queries are also adversarial ones!!!

Correlated queries



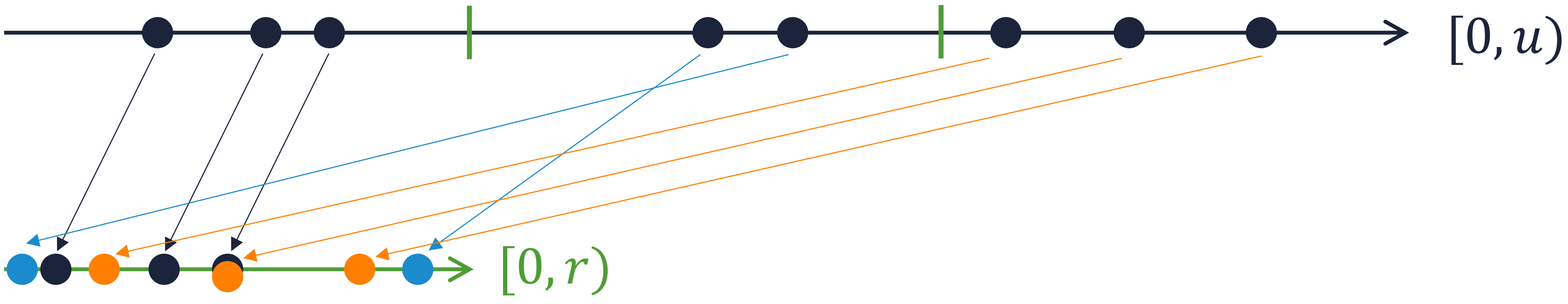
Grafite: hash

1. Apply the locality-preserving hash function by Goswami et al. [SODA ’15] to the input keys

2-independent hash function

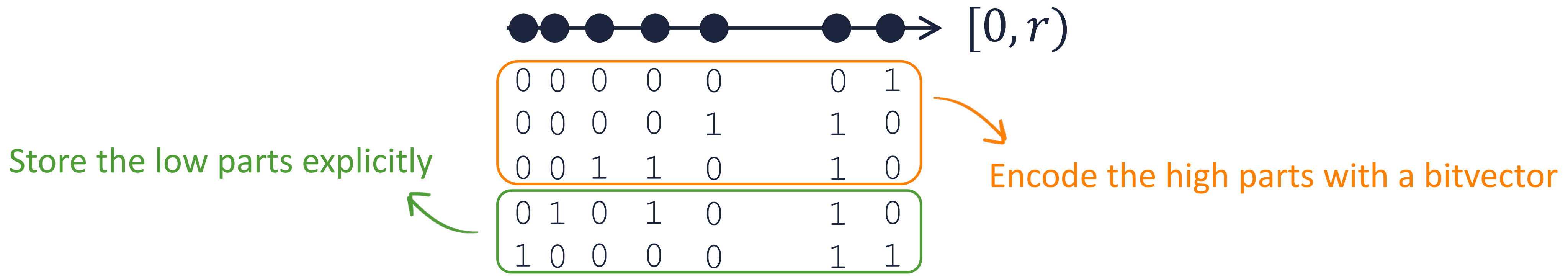
$$h(x) = \left(x + q(\lfloor x/r \rfloor)\right) \bmod r$$

Properly chosen to bound FPR and space



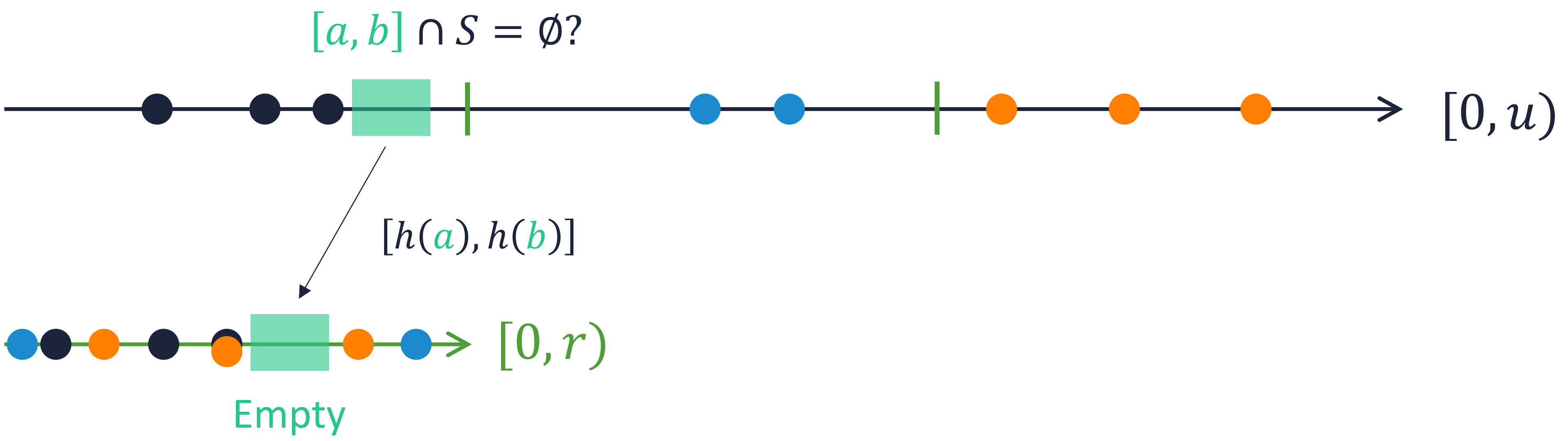
Grafite: hash + compress

2. Compress the hash codes with the Elias-Fano integer code



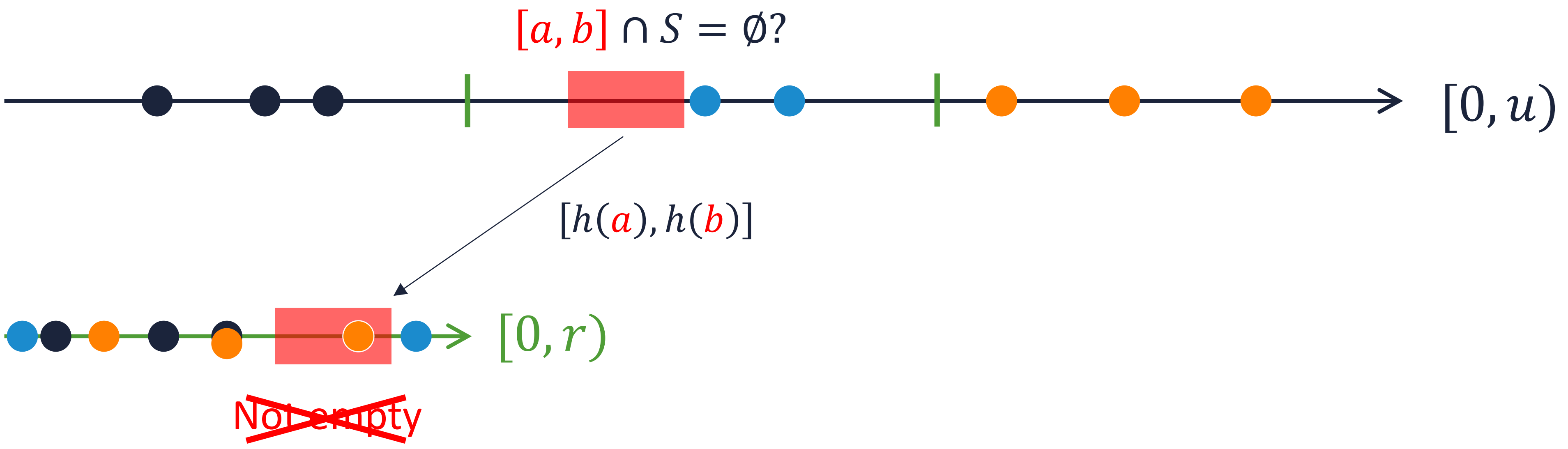
Grafite: hash + compress + query

3. Solve queries in hash space



Grafite: hash + compress + query

3. Solve queries in hash space



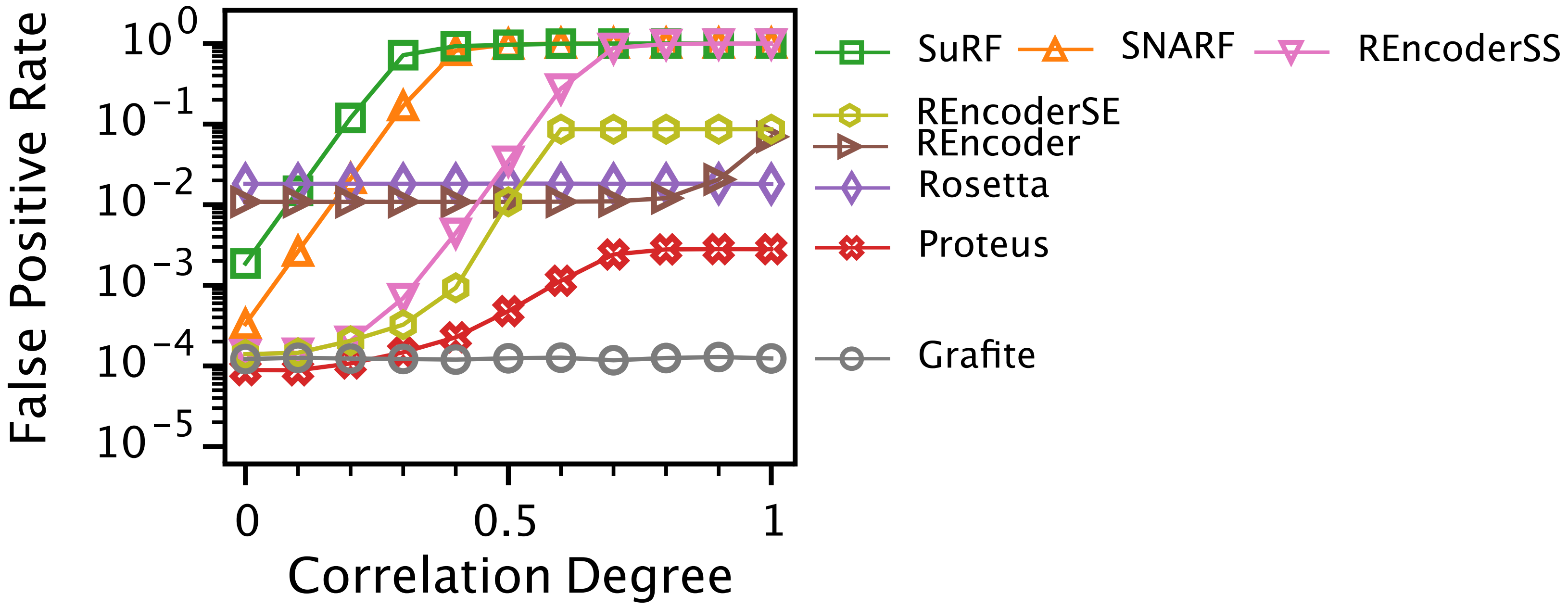
Grafite's theoretical guarantees

<u>Theorem</u>. Given a space budget of B bits/key, the **query time** is $O(1)$ and the **false positive probability** ε is $\leq \frac{\ell}{2^{B-2}}$, where ℓ is the query range size

- Optimal, according to the lower bound by Goswami et al. [SODA ’15]
- More succinct than their solution by more than 1 bit/key
- Works robustly out of the box; just specify B or ε

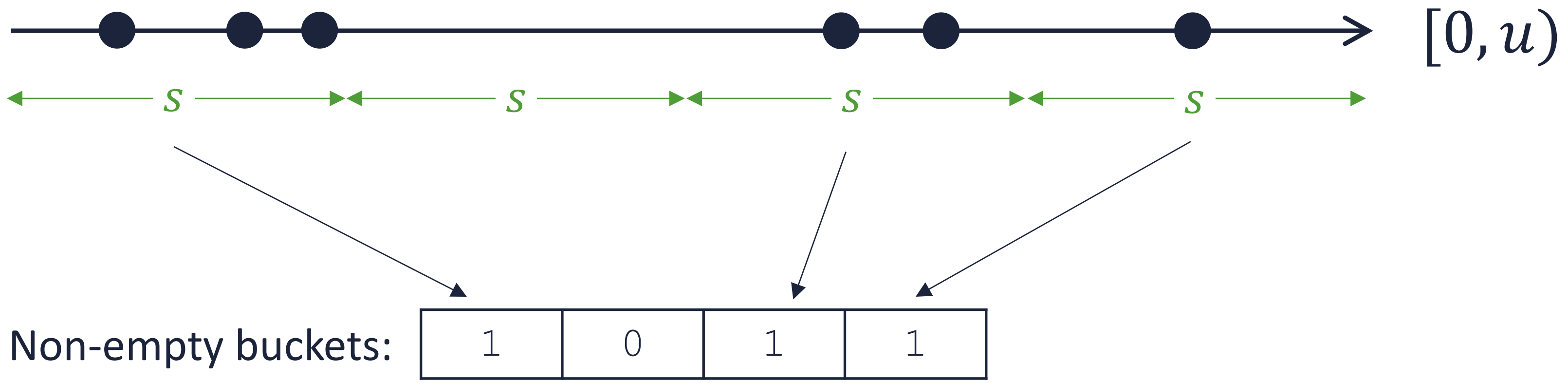
Robust vs heuristic range filters

- Unlike Grafite, most range filters are not robust:
  - Performance depends on the choice of dataset and queries
  - A malicious user can issue adversarial queries
  - Still, they could provide some benefit in non-adversarial cases
- Can we design simpler heuristic range filters?



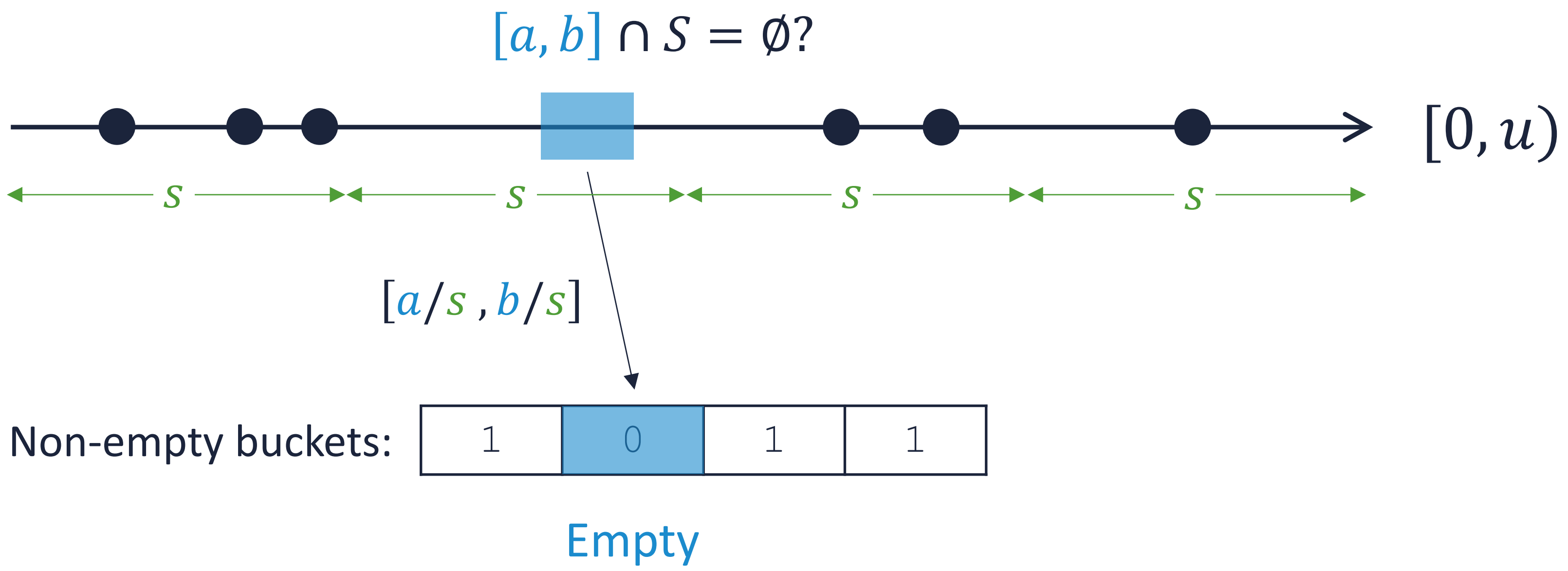
Bucketing: a simple heuristic range filter

1. Divide the universe $[0, u)$ into buckets of equal size s
2. Mark non-empty buckets with a compressed bit-vector
3. Solve queries by mapping ranges to bit-vector positions



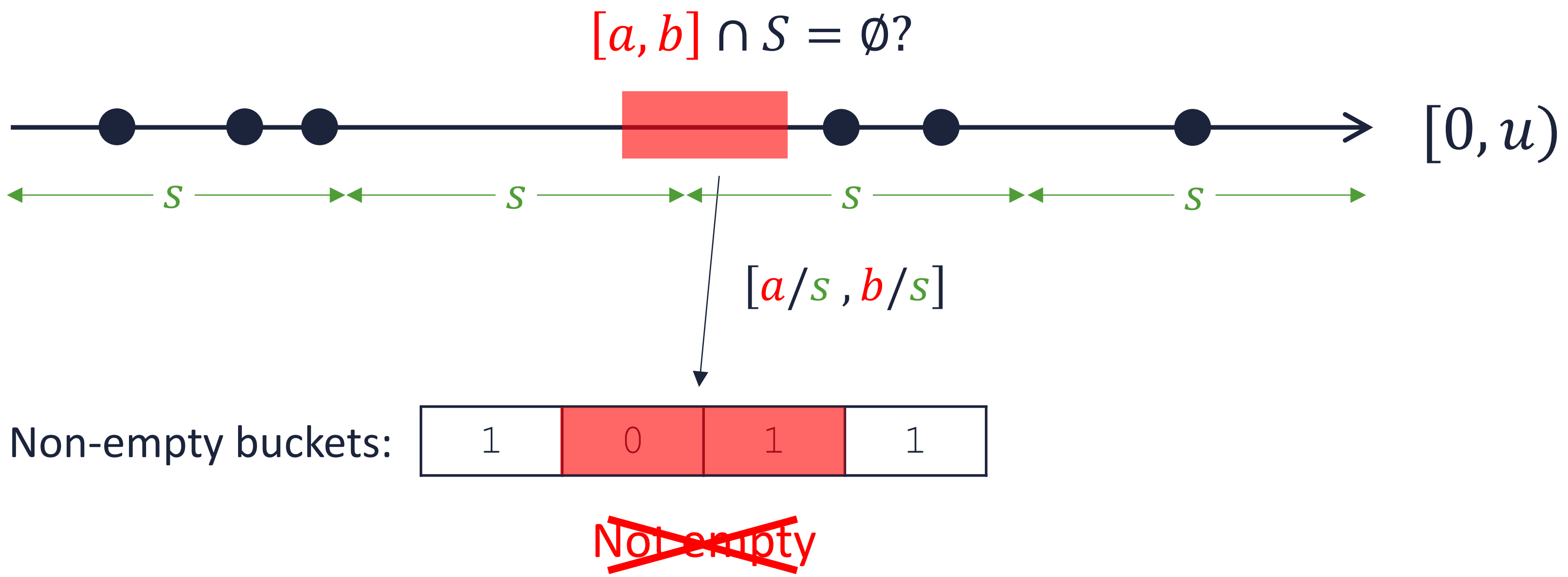
Bucketing: a simple heuristic range filter

1. Divide the universe $[0, u)$ into buckets of equal size s
2. Mark non-empty buckets with a compressed bit-vector
3. Solve queries by mapping ranges to bit-vector positions

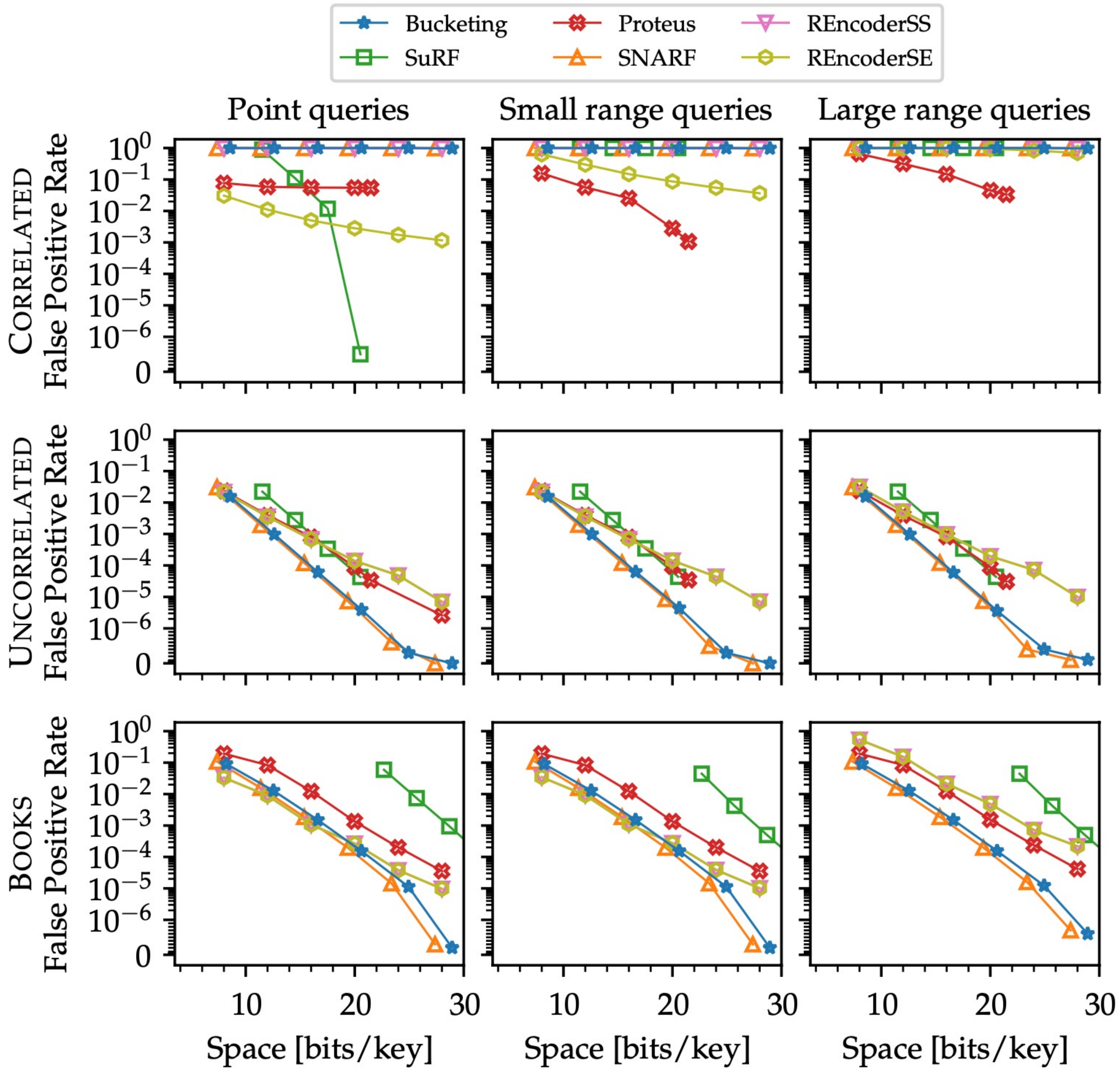


Bucketing: a simple heuristic range filter

1. Divide the universe $[0, u)$ into buckets of equal size s
2. Mark non-empty buckets with a compressed bit-vector
3. Solve queries by mapping ranges to bit-vector positions



Experiments with heuristic range filters



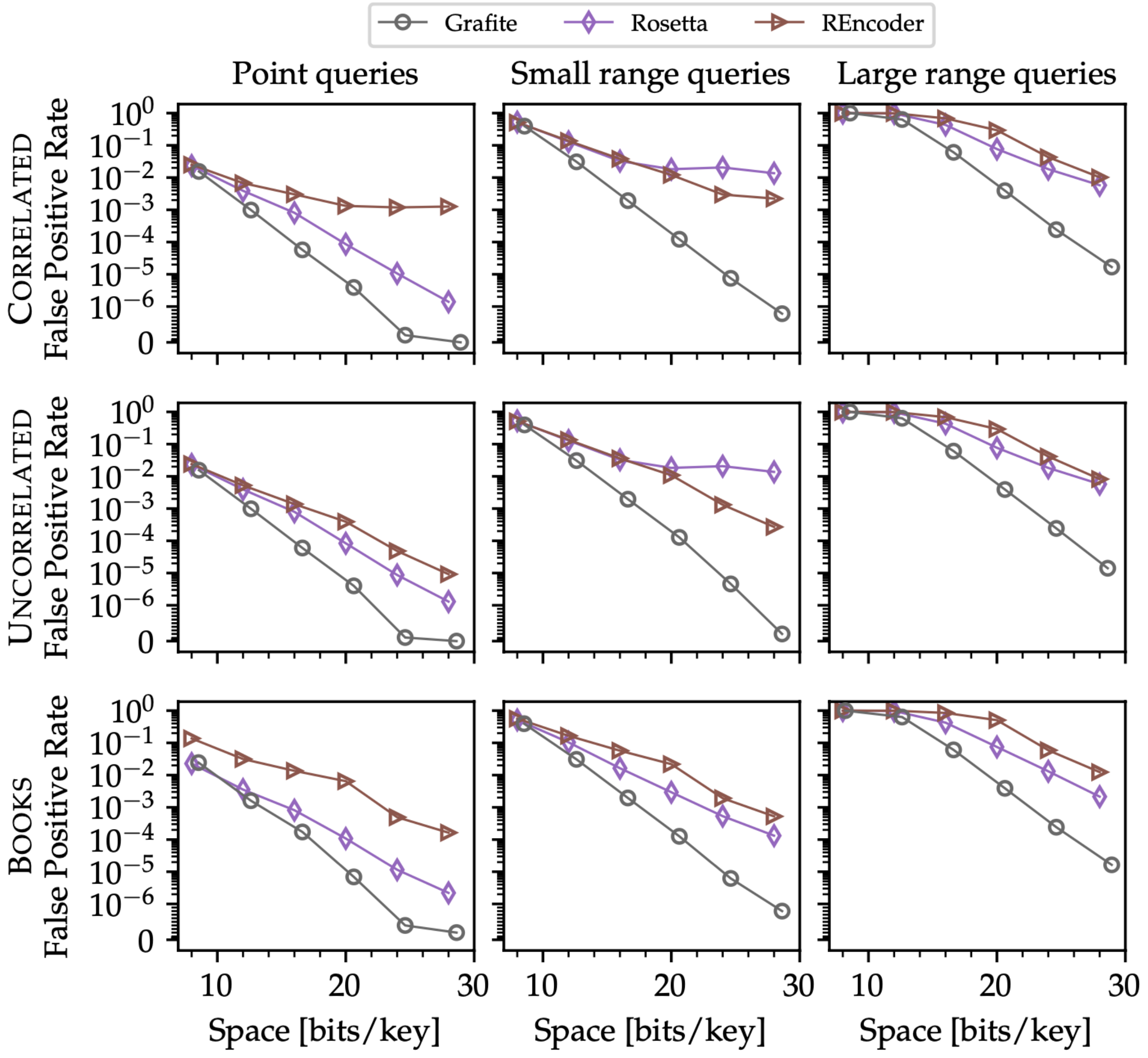
Range filter	Avg ns/query
Bucketing	67 (1.0×)
SuRF	275 (4.1×)
SNARF	576 (8.6×)
REncoderSS	732 (10.93×)
REncoderSE	4074 (60.81×)
Proteus	28638 (427.43×)

Bucketing	70 (1.0×)
Proteus	227 (3.24×)
SuRF	228 (3.26×)
REncoderSE	351 (5.01×)
REncoderSS	352 (5.03×)
SNARF	895 (12.79×)

Bucketing	277 (1.0×)
SuRF	423 (1.53×)
REncoderSS	527 (1.9×)
REncoderSE	528 (1.91×)
Proteus	573 (2.07×)
SNARF	1817 (6.56×)

- No filtering on CORRELATED queries, except for Proteus/REncoderSE:
  - Advantaged by auto-tuning
  - Very slow queries
- On other datasets, Bucketing offers:
  - FPR very close to the best-performing range filters
  - 6–13× faster queries
  - 5–7× faster construction
- **Take-home message**. Heuristic range filters sacrifice robustness to work well on specific inputs, but there might be simpler solutions like Bucketing

Experiments with robust range filters



Range filter	Avg ns/query
Grafite	305 (1.00×)
REncoder	2905 (9.52×)
Rosetta	24902 (81.65×)

Grafite	270 (1.00×)
REncoder	3001 (11.11×)
Rosetta	24913 (92.27×)

Grafite	274 (1.00×)
REncoder	2410 (8.80×)
Rosetta	24716 (90.20×)

- **Space-vs-FPR.** Grafite is up to 4 and 5 orders of magnitude more effective than REncoder and Rosetta, respectively
- **Query time.** Grafite is about 1 and 2 orders of magnitude faster than REncoder and Rosetta, respectively
- **Construction time.** Grafite is up to 8 and 10 orders of magnitude faster than REncoder and Rosetta, respectively
- **Take-home message.** If robustness guarantees are needed regardless of input data and future queries, Grafite is the range filter of choice

Conclusion

- **Grafite** solves the lack of robustness in current practical range filters
  - Bounded FPR in optimal space
  - Efficient and predictable performance

- **Bucketing** simplifies the design of heuristic range filters
  - Almost equivalent FPR
  - Faster query and construction times

- **Future work**
  - Engineer a version of Grafite for string keys
  - Support insertions and study their impact on the FPR