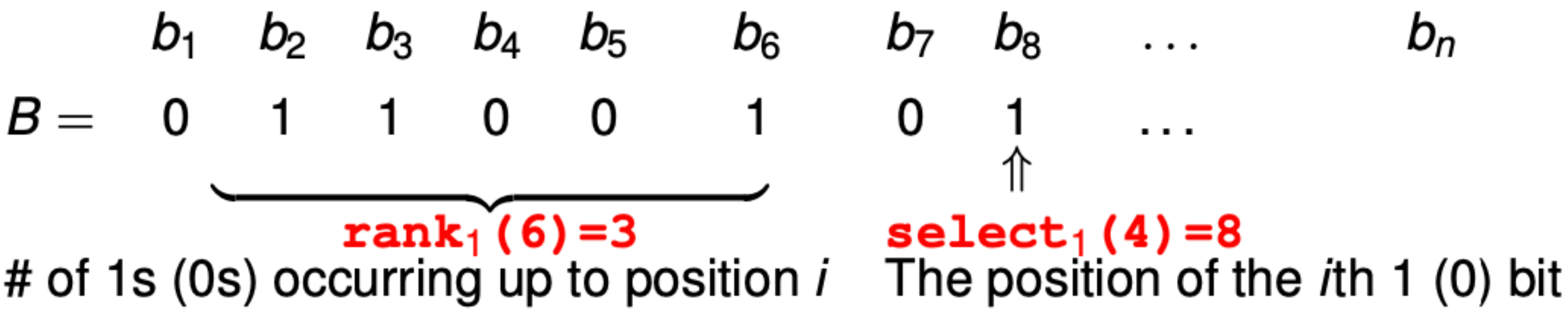
Counting with Prediction: Rank and Select Queries with Adjusted Anchoring

Oguzhan Kulekci
Indiana University Bloomington
okulekci@iu.edu

Data Compression Conference 2022

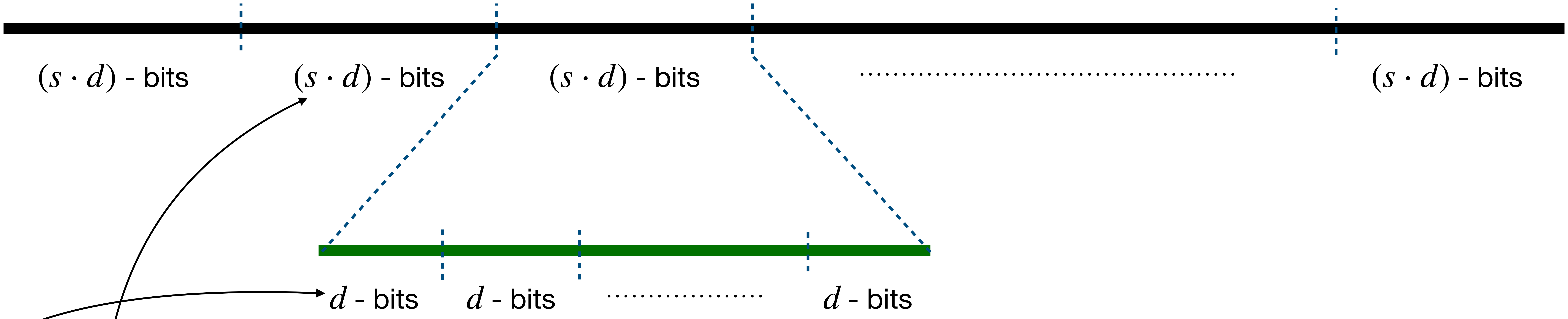
Rank & Select Queries



- The fundamental building block in compressed data structures.
- Deeply studied for more than 30 years (please see the references in the paper)

Rank&Select Dictionaries

Maintain a dictionary of rank values for some positions and use it to answer queries efficiently.



$n/(s\,.\,d)$ superblocks, each with $\log n$ bits **absolute** rank value

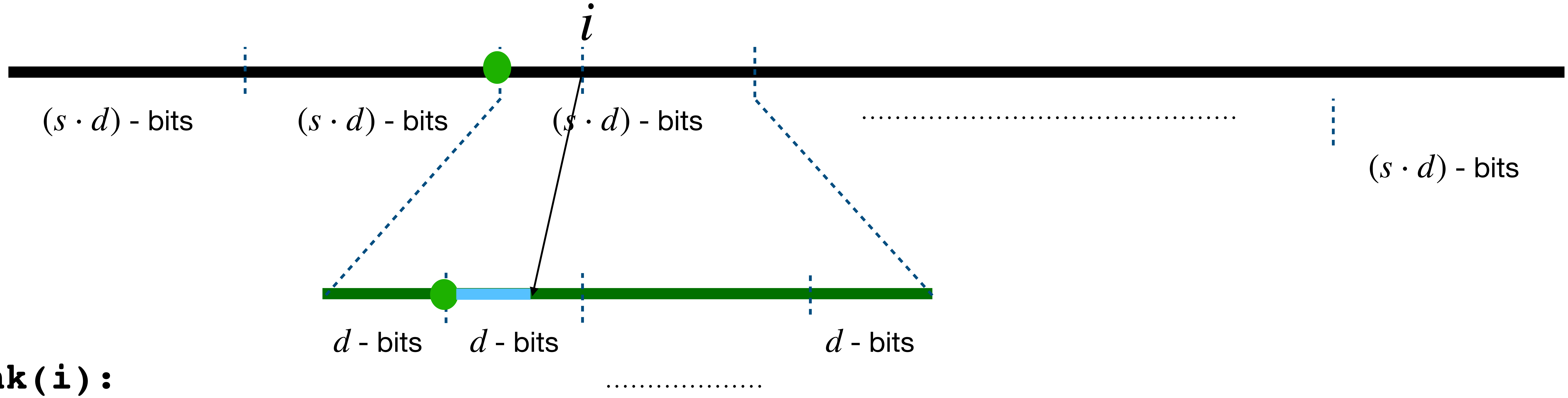
n/d blocks, each with $\log(s \cdot d)$ bits **relative** (to its superblock) rank value

Total overhead

$$n \cdot \left[\frac{\log(s \cdot d)}{d} + \frac{\log n}{s \cdot d}\right] = o(n)$$

,which becomes $o(n)$ with proper selection of s and d

Rank&Select Dictionaries



`Rank(i):`

- Sum the corresponding superblock and block rank values (green dot) from the maintained dictionary
- Add the number of set bits detected inside the inner-block up to the queried position (blue bar)
  - **SIMD instructions are used to compute this value fast in constant time.**

Overall, $O(1)$-time solution for rank with $o(n)$ overhead.

`Select(i):`

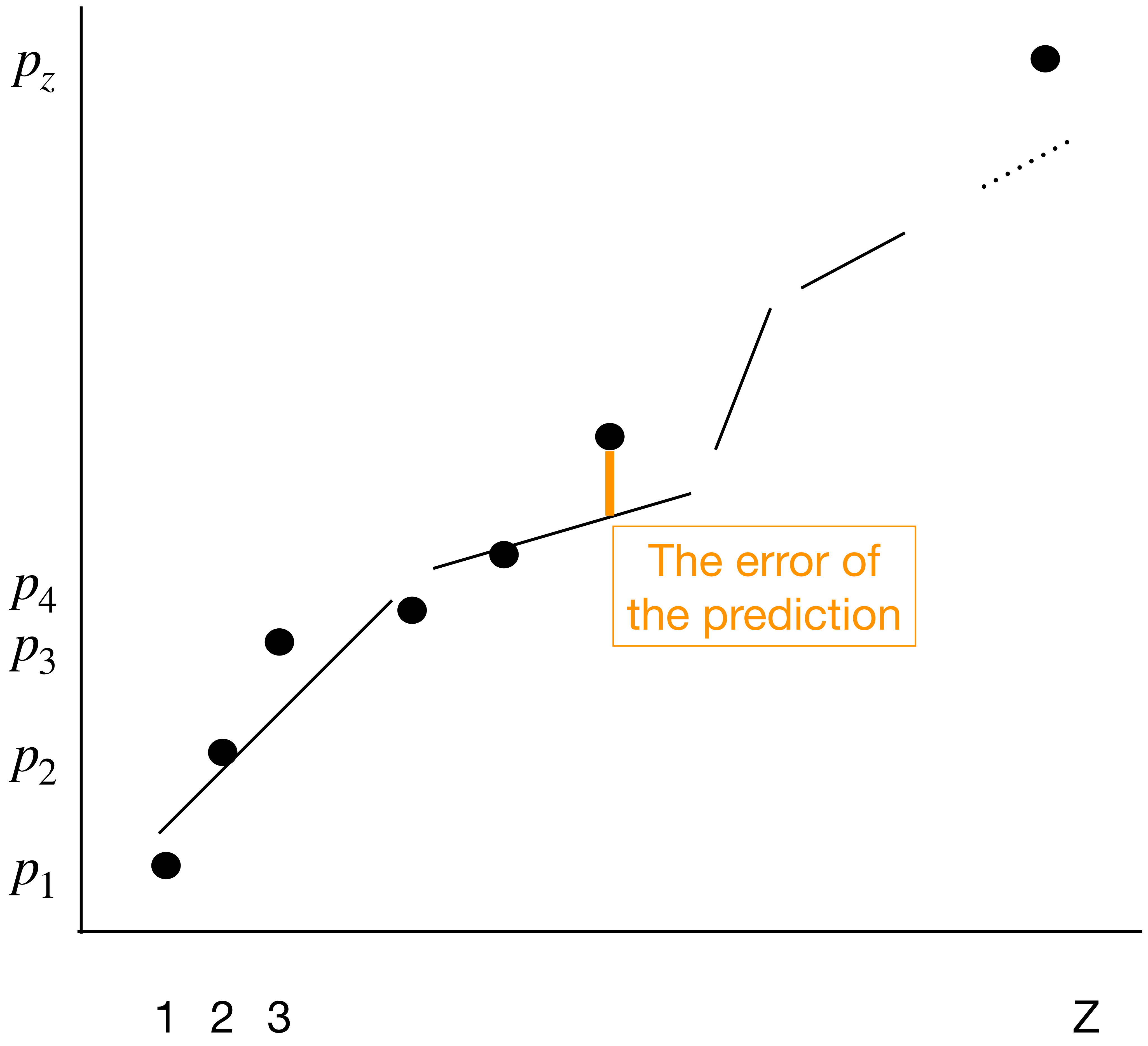
- Yes, it needs a different data structure with variable size blocks to answer in constant time.

Rank&Select Dictionaries

- In practice, the data structures **favor either rank or select** operations, but not both ! Usually, the dictionaries constructed with fixed-size blocks favor rank, and variable-size ones select.
- If the bitmap is sparse (the polarity is far from 0.5) then keeping the bitmap compressed make sense, and leads to **compressed R&S solutions**. (Although they are a bit slow in practice still)

- New research direction by **using machine learning techniques in data structure design**, *Learned data structures, Ferragina, Vinviguerra, 2020.*
- ***Boffa et al, ALENEX'21:*** *A learned-approach to quicken and compress R&S Dictionaries*
  - *Targets* ***compressed*** *bitmap, favors* **`SELECT`** *with variable size blocks*
- ***This study***
  - *Targets* ***uncompressed*** *bitmaps, and favors* **`RANK`** *with fixed size blocks*

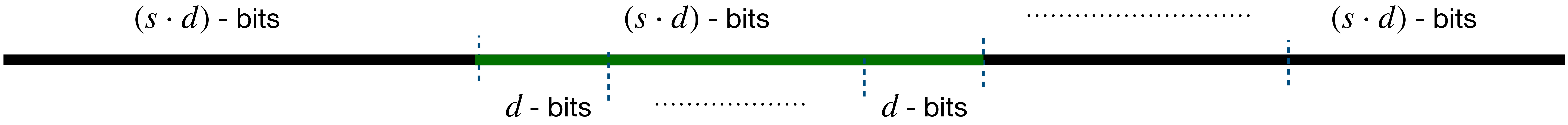
Previous Work: R&S with Learning Approach



- The **positions of the set bits** on a given bitmap is a sequence of **increasing** integers $P = \langle p_1, p_2, \ldots, p_z \rangle$
- Fit ℓ lines, where each covers variable number of positions such that the error between the prediction and actual value is denotable by c bits.
- Maintain the parameters for each $ax + b$ line, and also the c bit correction values for each position
- Also some metadata for the number of positions covered per each line is stored
- **The ℓ depends on the regularity and number of the positions**

- **Select(i):** Simply go the the line corresponding line, get the prediction and correct it.
- **Rank(i):** Search which line includes I and search the closest previous position on that line.

Proposed Data Structure



- The block rank values in each super-block is an increasing sequence
  - $B_1 = \langle b_1^1, b_2^1, \ldots b_{s-1}^1 \rangle,\quad \ldots\ldots,$
    $B_{n/sd} = \langle b_1^{n/sd}, b_2^{n/d}, \ldots b_{s-1}^{n/sd} \rangle$
  - Linear regression $(ax + b)$ per each B_i and **store** the (a, b) values that occupies $\dfrac{n}{s \cdot d} \cdot (32 + 32)$ bits.
- Per each block maintain **a single validity** bit and a $\log m$ bit **correction value** that consumes $\dfrac{n}{d} \cdot (1 + \log m)$ bits.

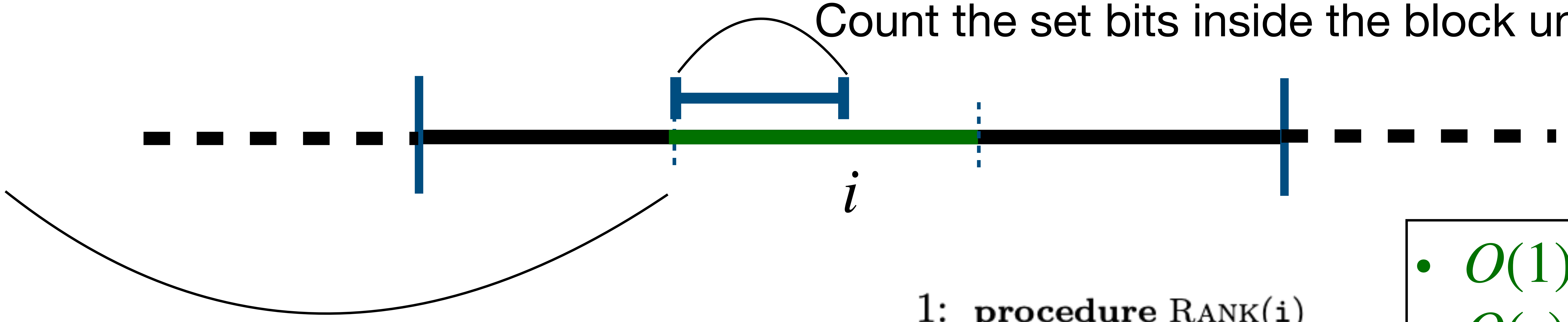
Total space usage in bits

$$\frac{n}{d} \cdot \left(\log 2m + \frac{64}{s}\right)$$

- **Nothing is stored for the n/sd super-block rank sequence !**
- **They are *almost* encoded with the b parameters of the regression lines $ax + b$, when $x = 0$**

Rank with Adjusted Anchoring

Count the set bits inside the block until position i via popcount.



- $O(1)$-time, without scan
- $O(s)$-time, if scan is required

- **Predict** the count of the set bits until the beginning of the block by using the corresponding regression line
- **Adjust** predicted value by using the corresponding correction value.

- If the error is larger than the adjustable range, then **scan towards left** until hitting a correctly predictable block.

```
procedure Rank(i)
    bID ← ⌊(i + d − 1)/d⌋
    excess ← popcnt(B[i + 1 . . . bID · d])
    rnk ← 0
    while (V[bID]=1)&(R[bID]=0)&(bID>0) do
        rnk ← rnk+popcnt(B[(bID−1)·d+1..bID·d])
        bID ← bID − 1
    if (bID > 0) then
        sID ← ⌊(bID + s − 1)/s⌋
        innerbID ← i − (bID − 1) · d
        rnk ← rnk + ⌊β_0^sID + β_1^sID · innerbID⌋
        rnk ← rnk − m/2 + R[bID]
        if V[bID] = 1 then
            if R[bID] ≥ m/2 then
                rnk ← rnk + m/2
            else
                rnk ← rnk − m/2
    return (rnk − excess)
```

Select with Adjusted Anchoring

```
procedure SELECT(i)
    q ← ⌊i/setBitRatio⌋
    sID ← ⌊(q + s·d − 1)/(s·d)⌋
    while (β_0^sID ≥ i)&(sID > 0) do sID ← sID−1
    while (β_0^sID < i) do sID ← sID+1
    ----------------------------------------
    innerbID ← ⌊(i−β_0^sID)/β_1^sID⌋
    if innerbID < 1 then innerbID = 1
    bID ← innerbID + sID · s
    if bID > sbc then
        bID←sbc
        innerbID←(bID mod s) + 1
    ----------------------------------------
    rank ← RANK(bID · d)
    c ← popcnt(B[(bID−1)·d+1..bID·d])
    while rank < i do
        bID ← bID + 1
        c ← popcnt(B[(bID−1)·d+1..bID·d])
        rank ← rank + c
    rank ← rank − c
    while rank ≥ i do
        bID ← bID − 1
        c ← popcnt(B[(bID−1)·d+1..bID·d])
        rank ← rank − c
    ----------------------------------------
    p ← KTHSETBIT_SIMD(i − rank)
    return (bID−1)·d + p
```

- Start with a rough **prediction** of the super-block according to *average set bit ratio*
- Perform a **linear scan** to locate it explicitly

- By using the corresponding linear regression, **predict** the block position inside the super block

- The predicted block ID is adjusted by checking its rank value
- In case it is needed, again the neighboring blocks are scanned towards left or right until the correct block is located.

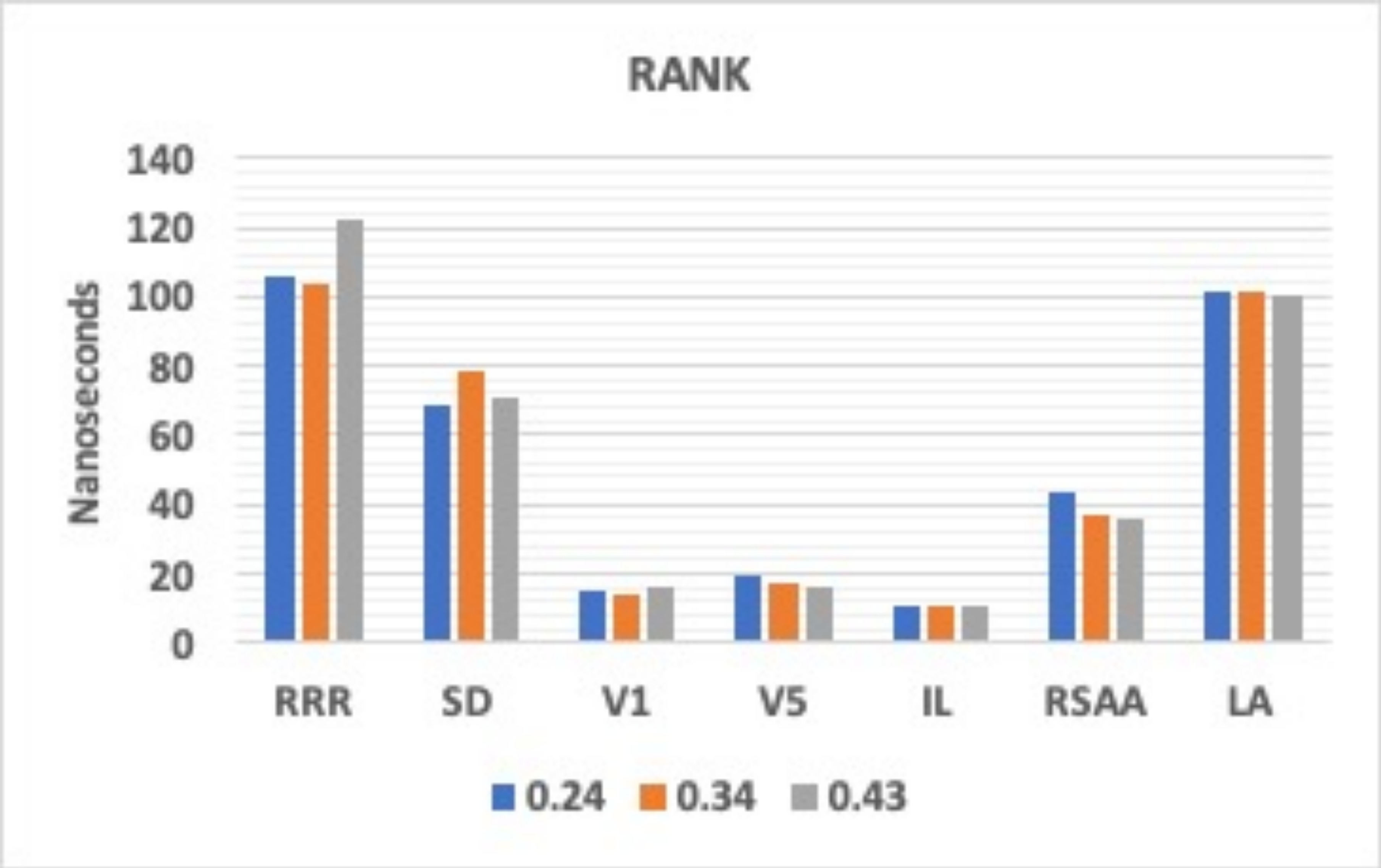
- Last but not the least, the kth set bit is determined with **SIMD instructions**.

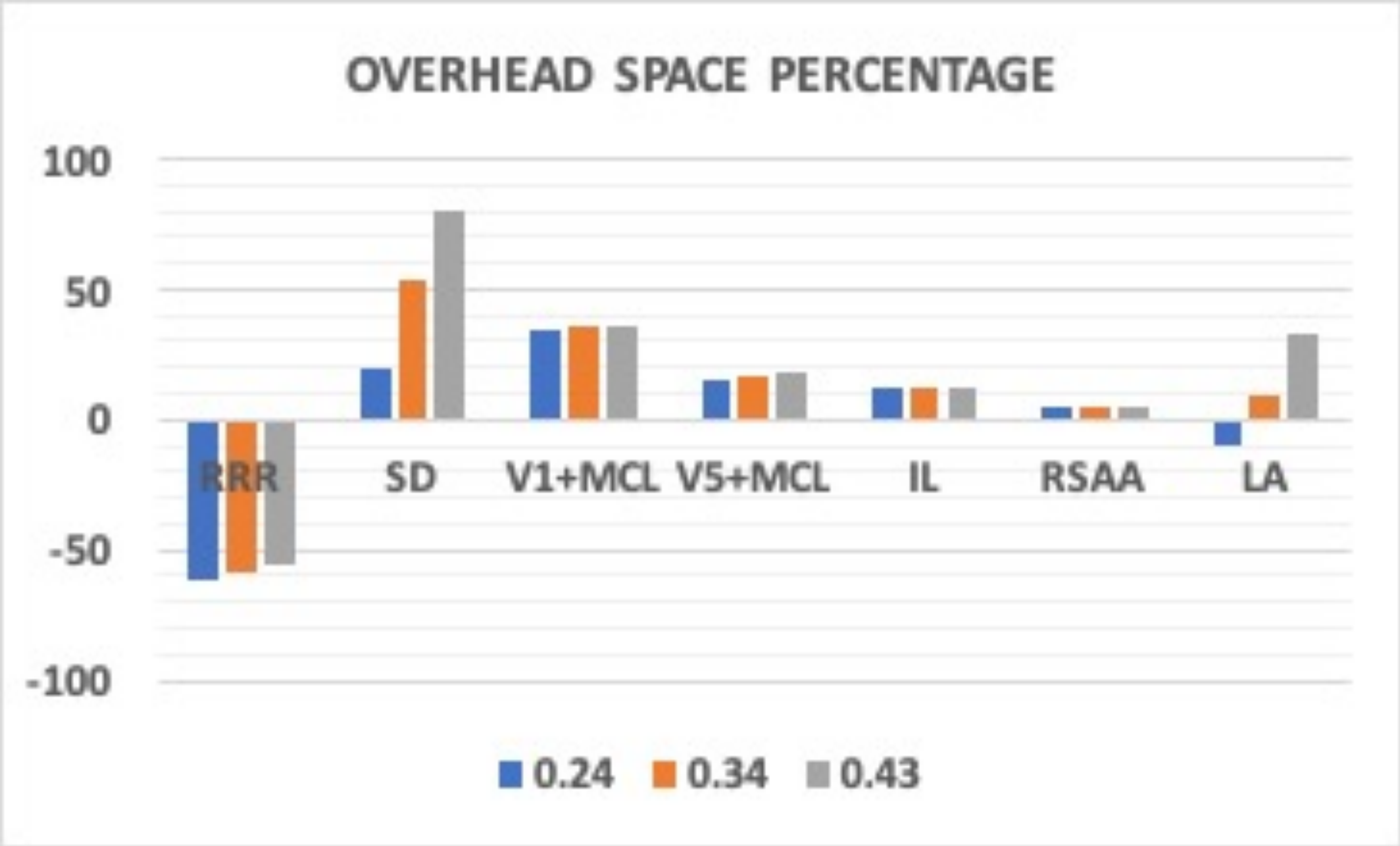
Tuning the parameters

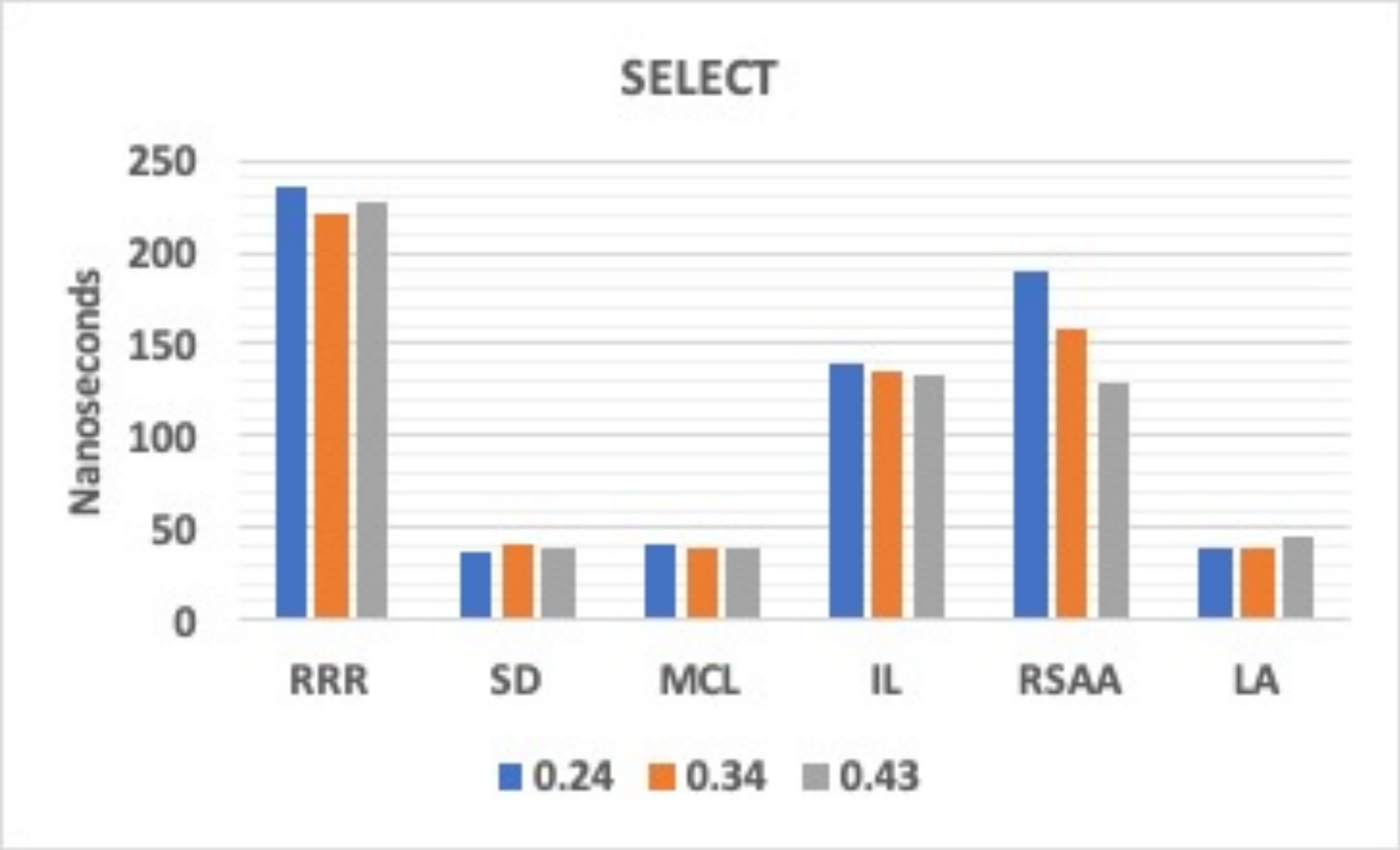
- **The overhead space is less than 5 percent, and correct prediction of the block rank values is more than 99% .**

- The performance of the RSAA scheme highly depends on accurate predictions, where the parameters s, d, m are central in prediction performance.
- Setting $m = d = 256,\ s = 16$ has been observed to provide most reasonable results empirically.
- Larger m, which denotes the recoverable error threshold, results in improved prediction success, but increases space usage (*and vice versa for sure*). For fast processing with small space consumption, $m = 256$ has been the best value.
- d, the block size in bits, is set to 256 as well to keep space consumption less than $5\,\%$ while not hurting the prediction performance with $m = 256$.
- To keep the irregular block count less than $1\,\%$, s, the number of blocks in a super block is set to 16.

Experimental Results on Real-Data

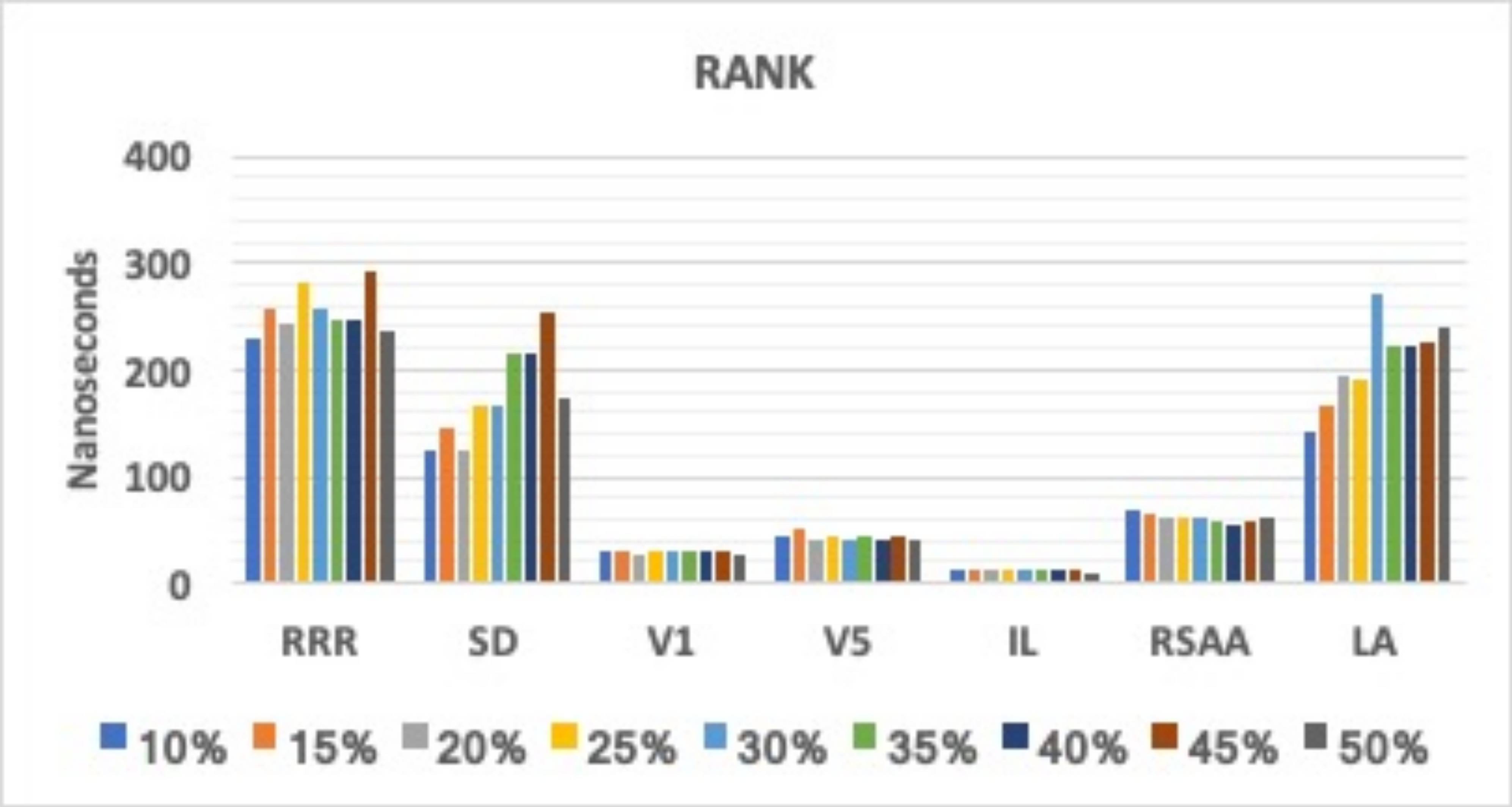


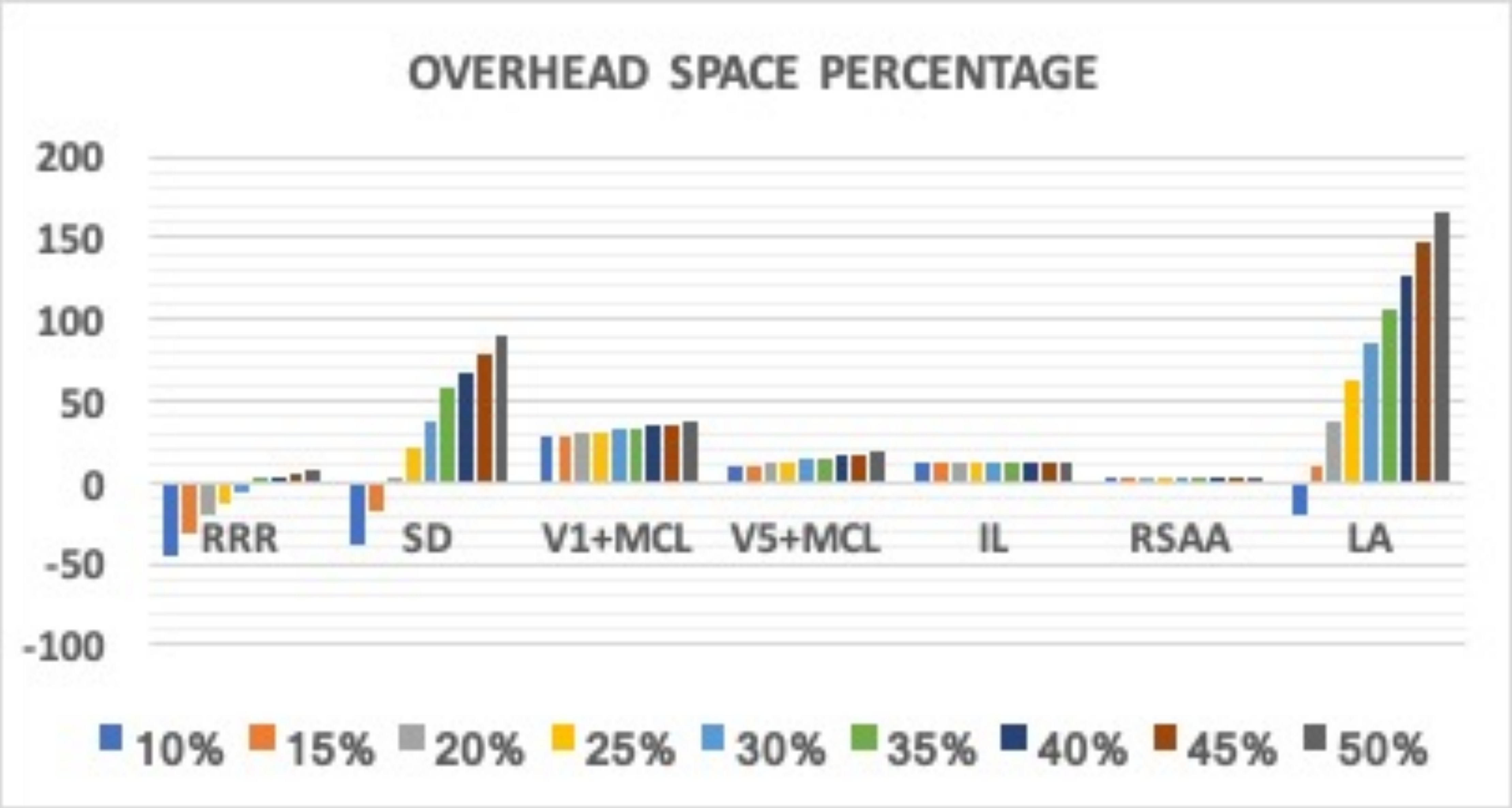


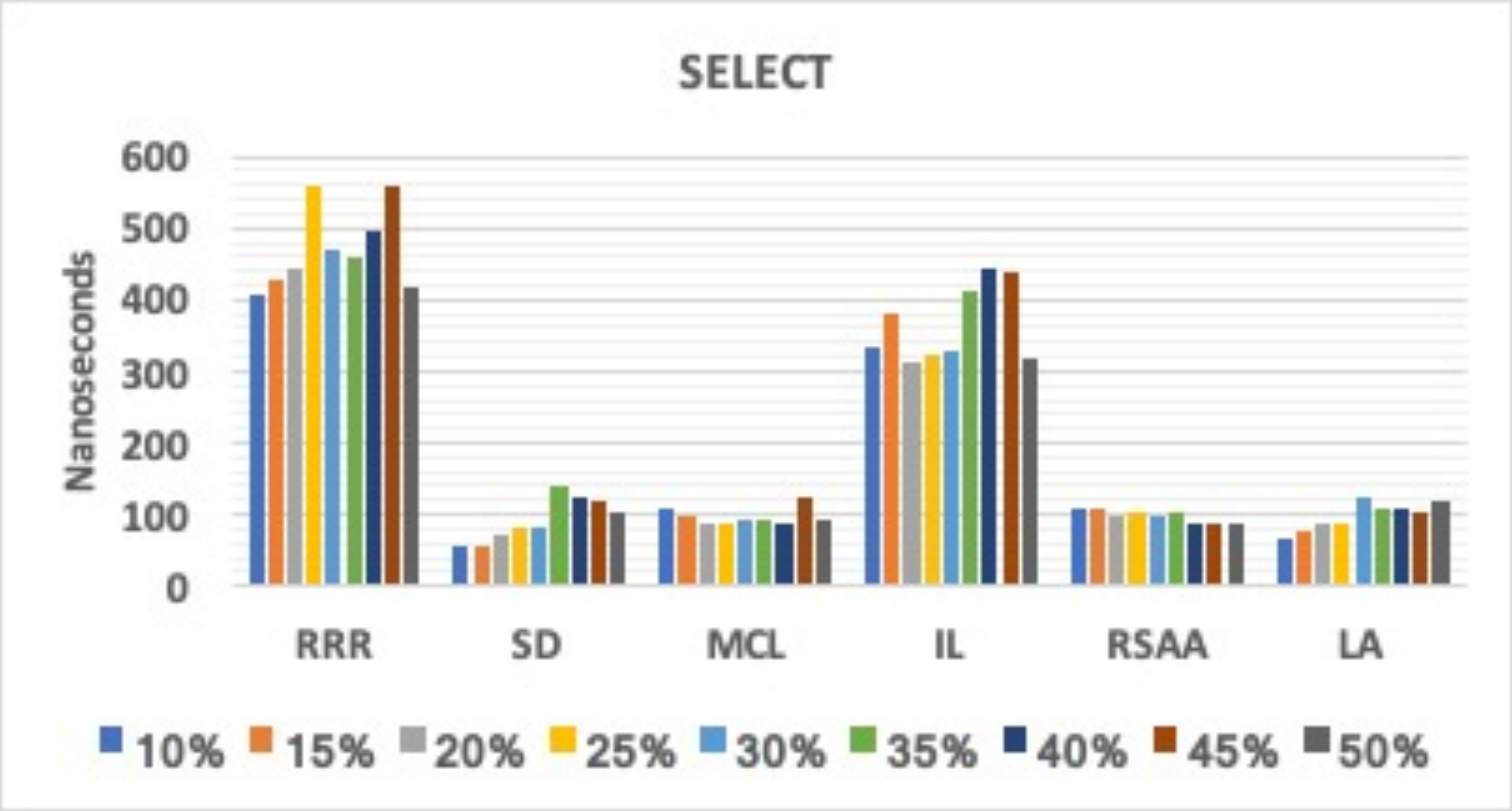


- RRR,SD, LA are compressed, V1,V5,MCL are uncompressed schemes
- Data sets are the real sequences used in previous study, which are averaged according to 0-1 ratios as 24, 34, and 43 percent.
- RSAA has the least overhead space
- LA compressed only the sequences with less than 30 percent set bits!

Experimental Results on Synthetic-Data







- On randomly generated bit sequences with different densities
- RSAA has the least overhead space
- Randomly distributed, balanced 0-1 distribution favors RSAA.

Conclusions

- Rank and select on **uncompressed bitmaps** with the machine learning support
- Studies appeared targeting sparse bitmaps previously, where, in contrast, RSAA targets **balanced density** bitmaps
- Overhead around 3-5% of the input
- **Would it be possible to have better time-space resuşts with other learning paradigms ?**
- **More generally, can ML techniques help in basic combinatorial tasks ?**