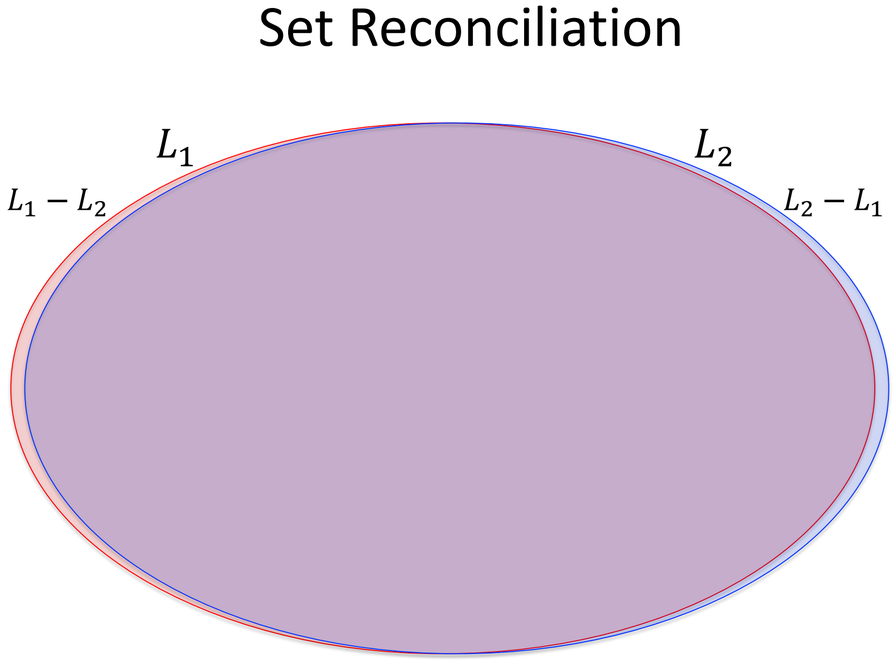
Invertible Bloom Lookup Tables

Lucas Kellar - 12/04/2024

The Set Reconciliation Problem

- Extract differences between very large sets with high overlap
- Datasets that hold small amount of data at extraction time, but large amounts before then

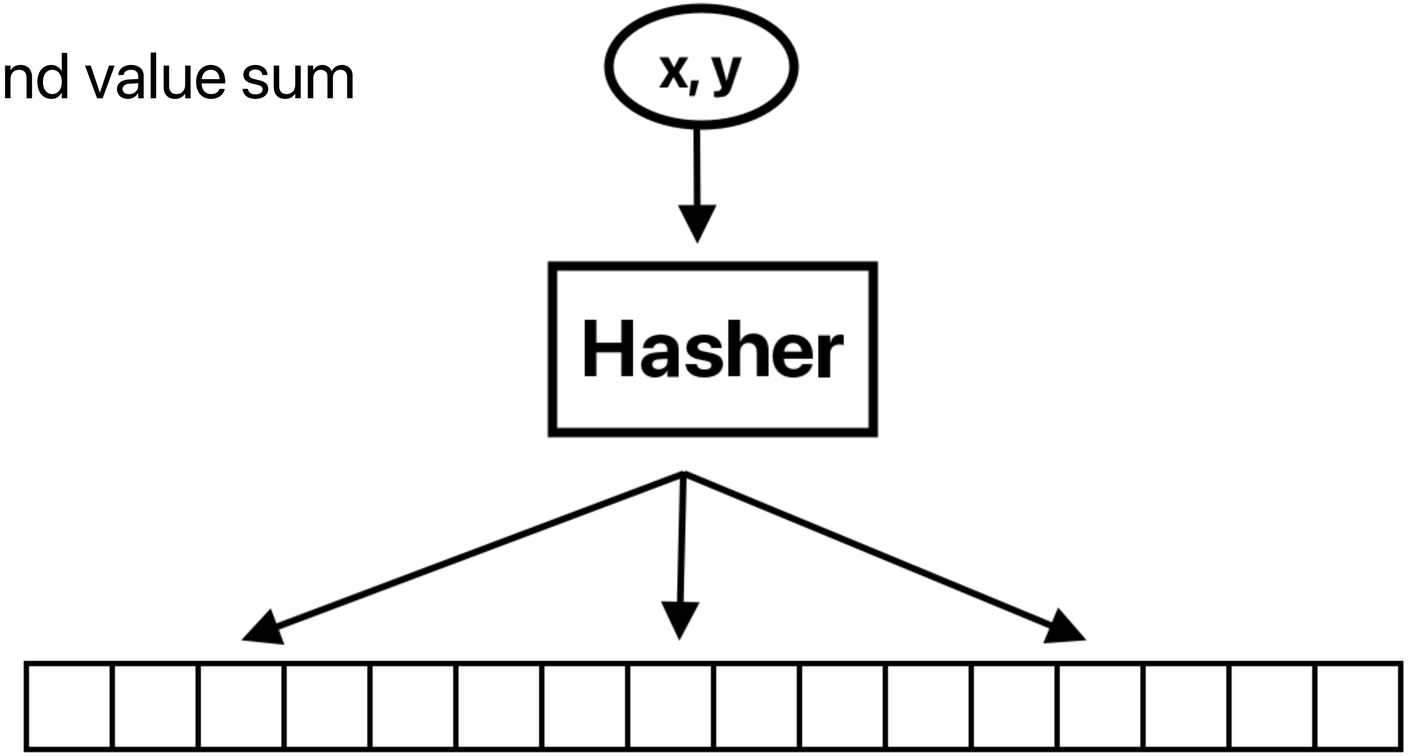


Implementation

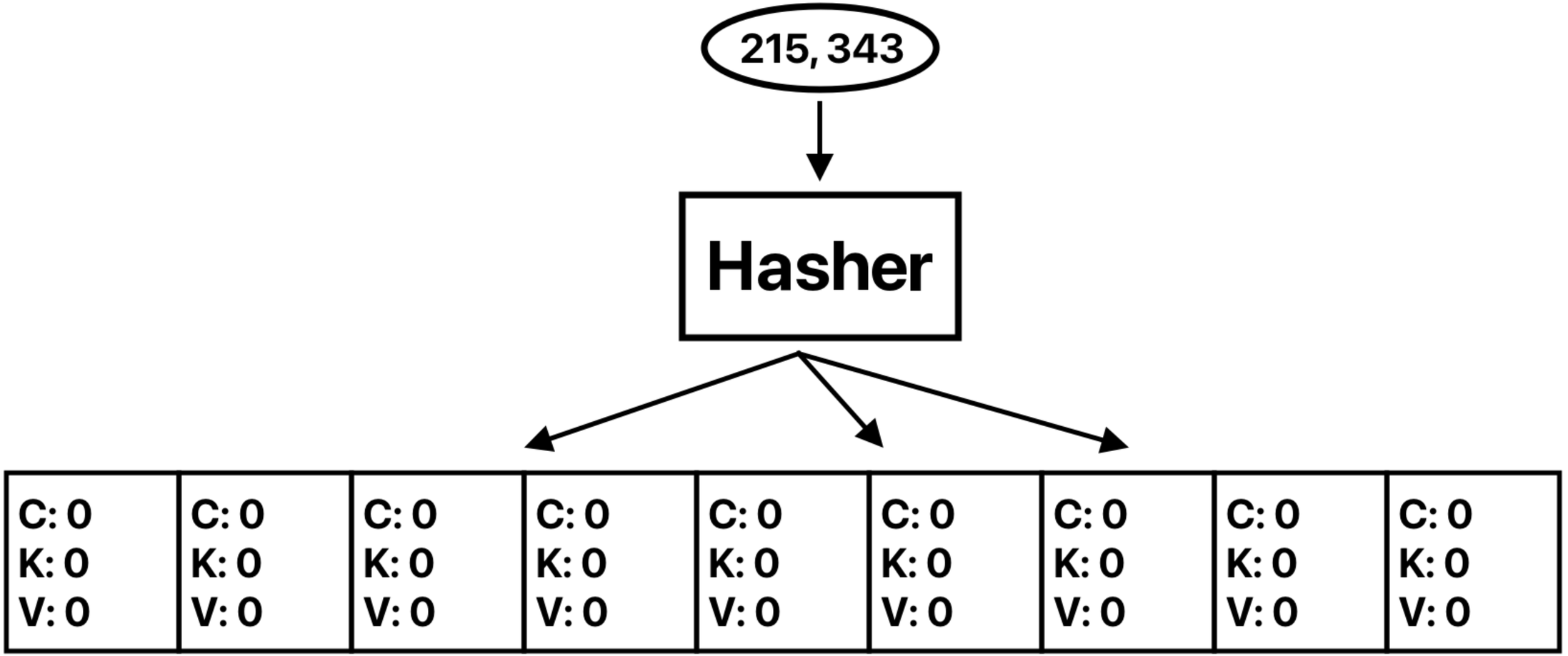
- Original IBLT (2011)
- HPW IBLT (2023)
- Stacked IBLT (2024)

Original IBLT

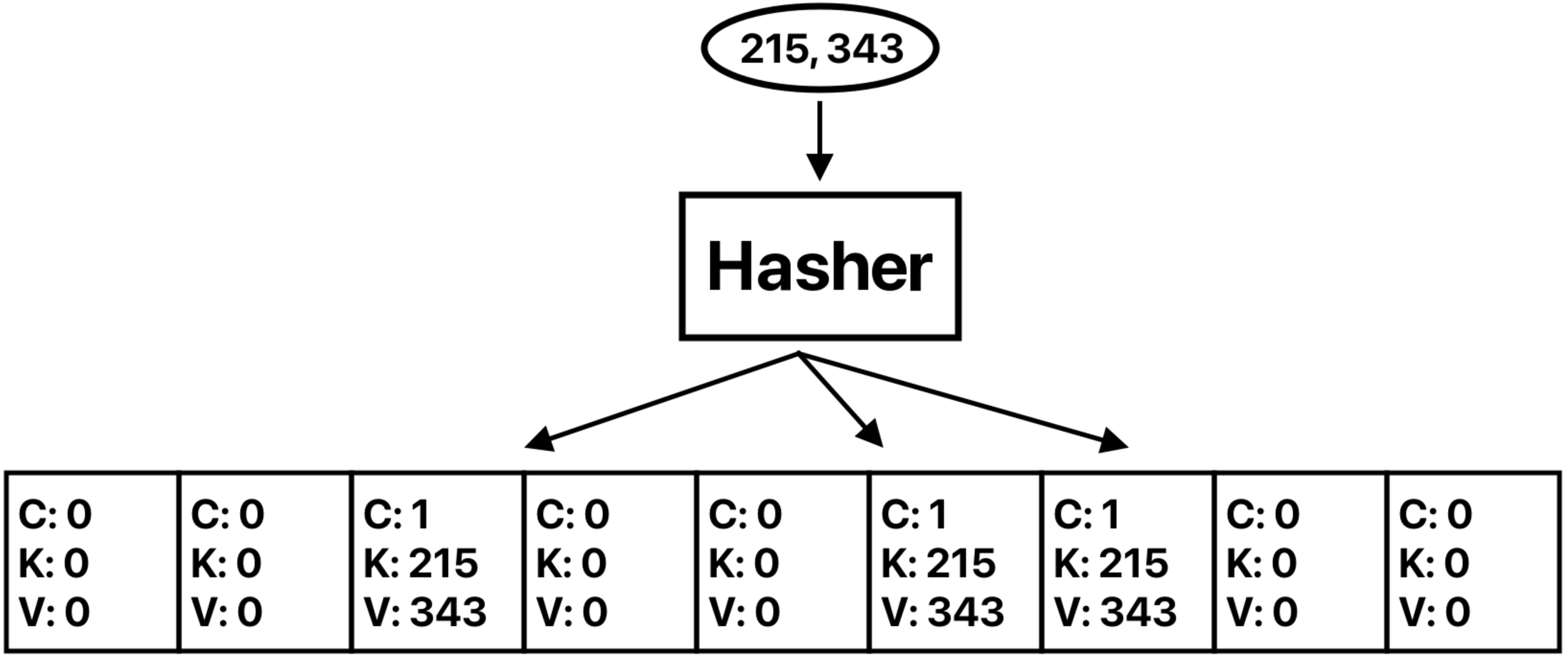
- Key/Value Store
- Each cell stores count, key sum, and value sum
- Operations:
  - Insert(x, y)
  - Delete(x, y)
  - Get(x)
  - ListEntries()



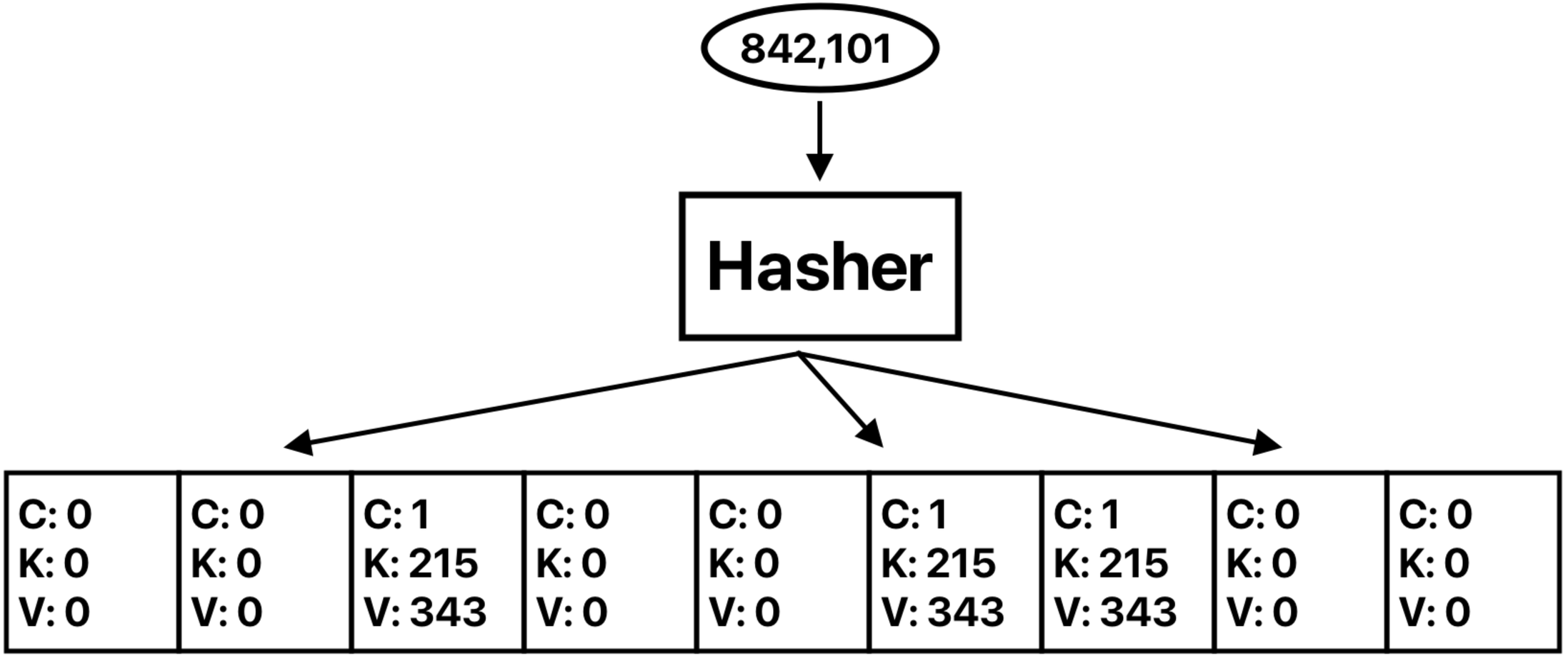
Original IBLT - Insert(x, y)



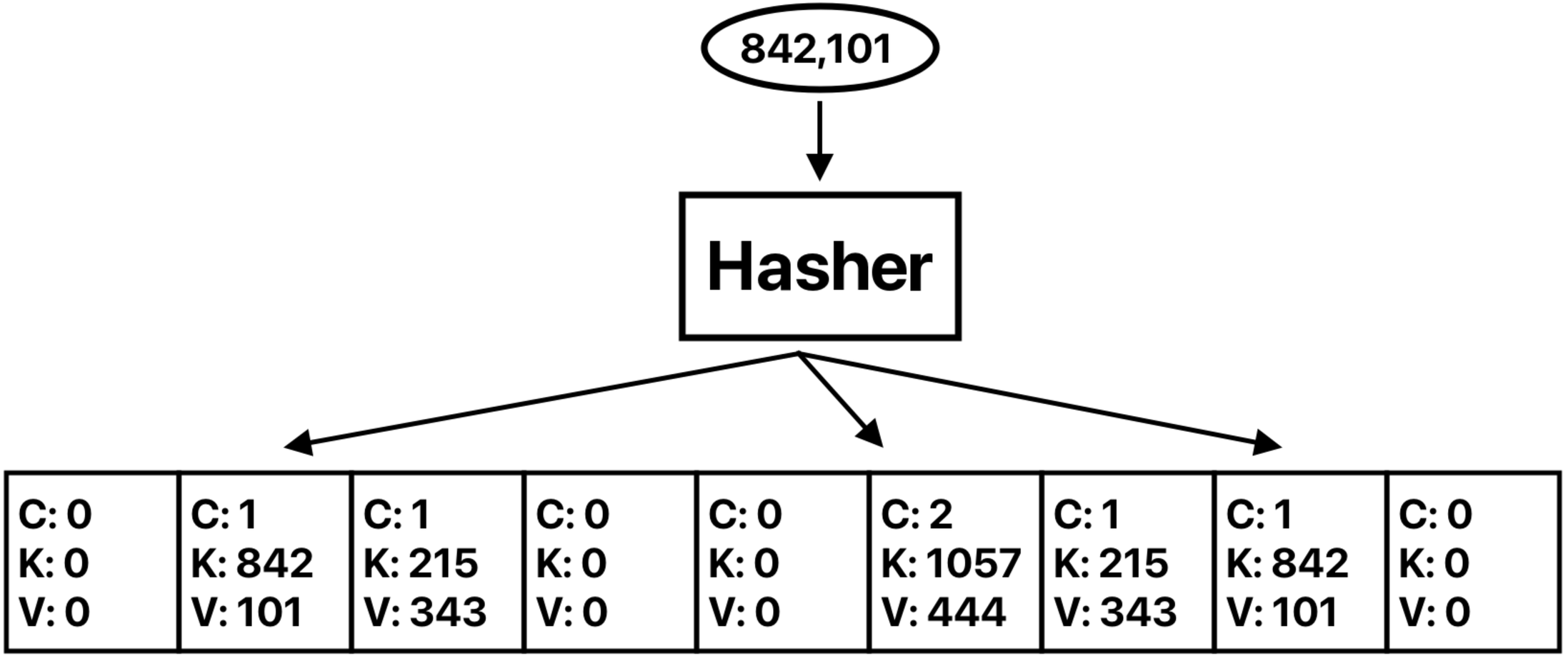
Original IBLT - Insert(x, y)



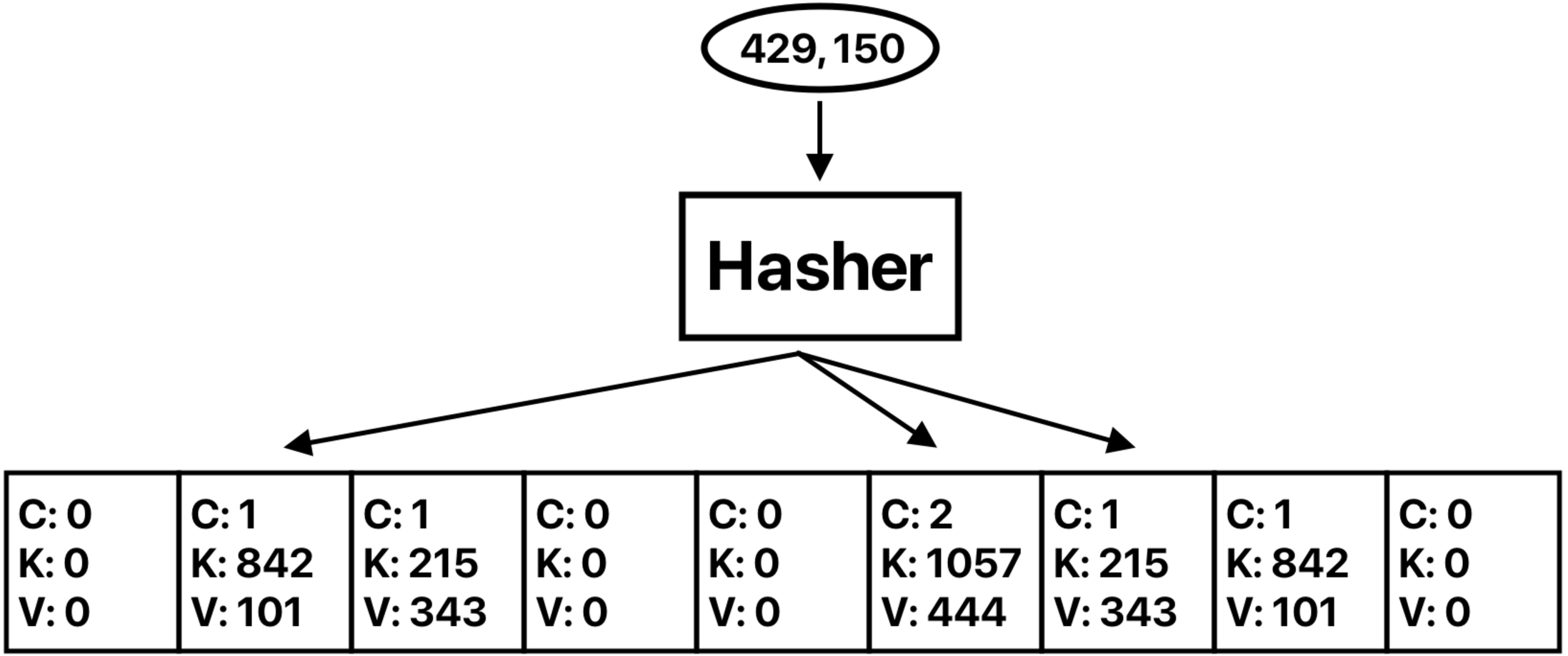
Original IBLT - Insert(x, y)



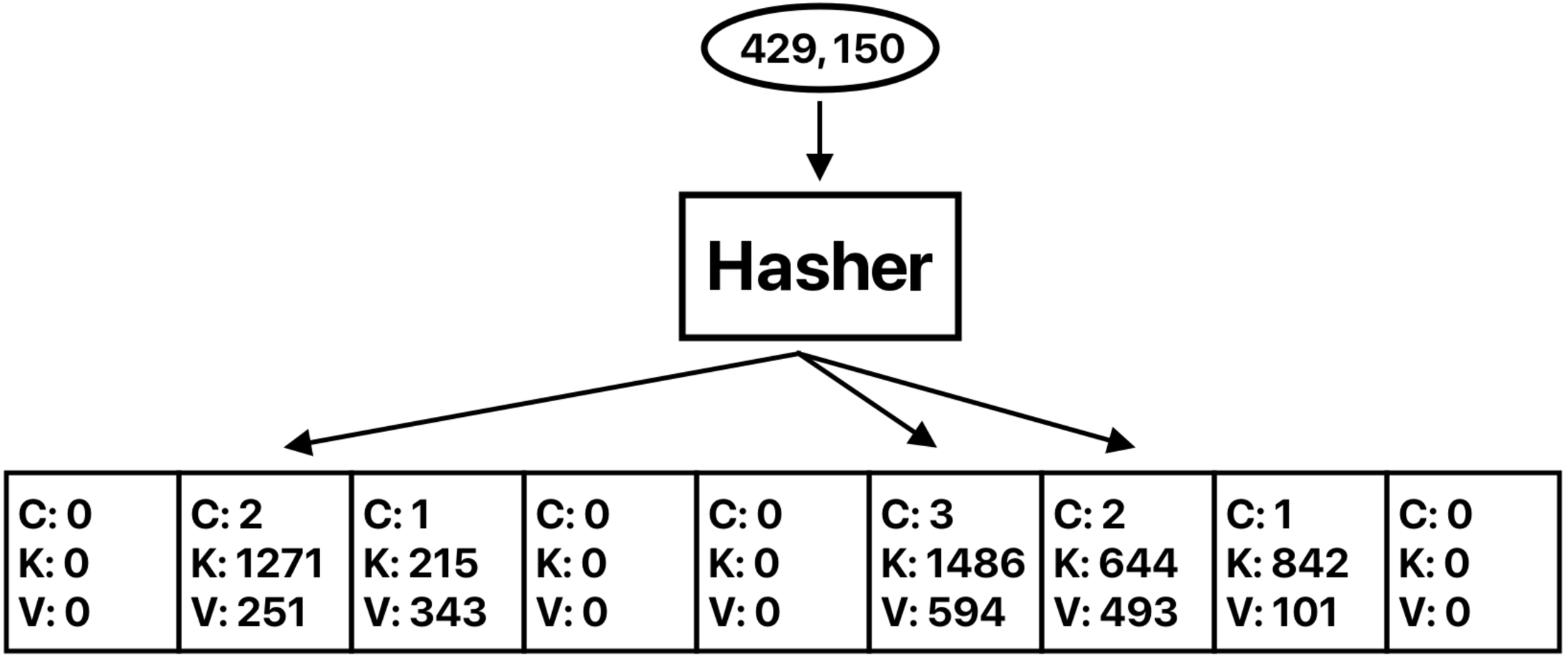
Original IBLT - Insert(x, y)



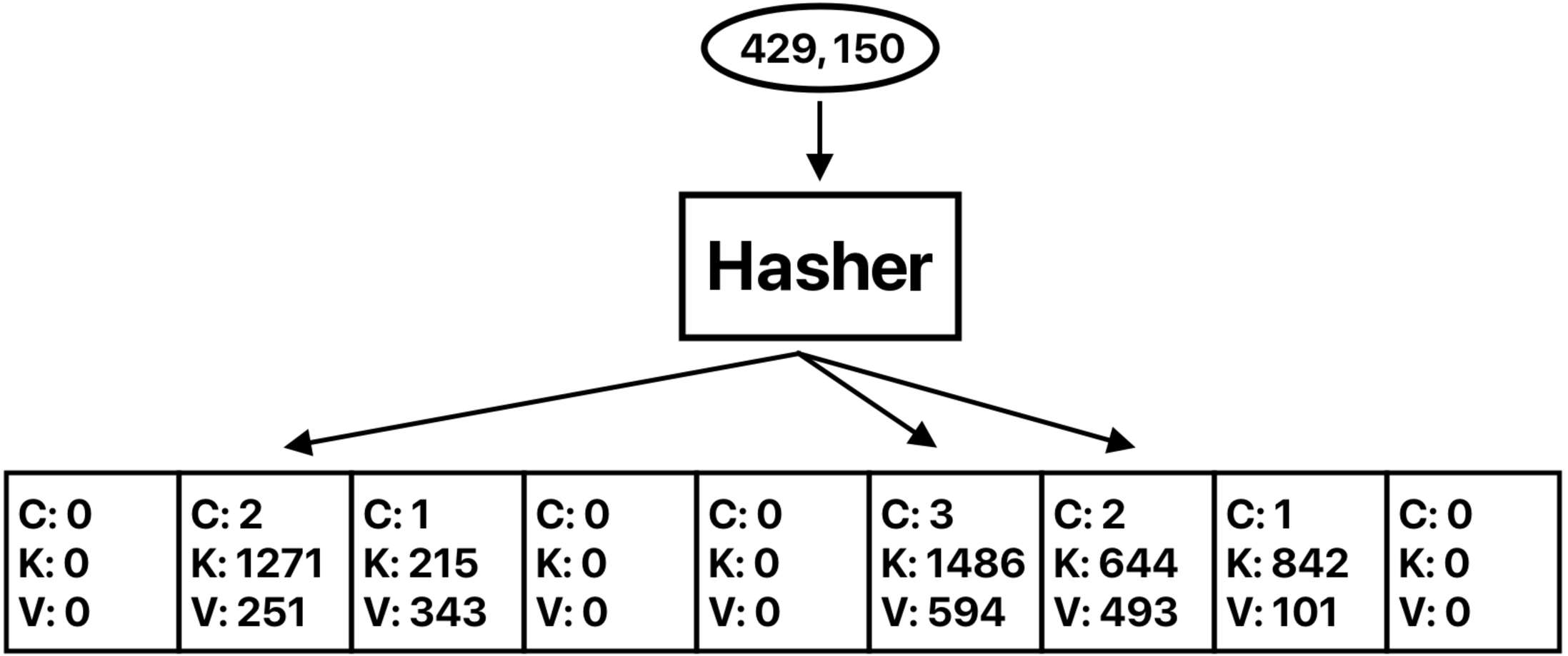
Original IBLT - Insert(x, y)



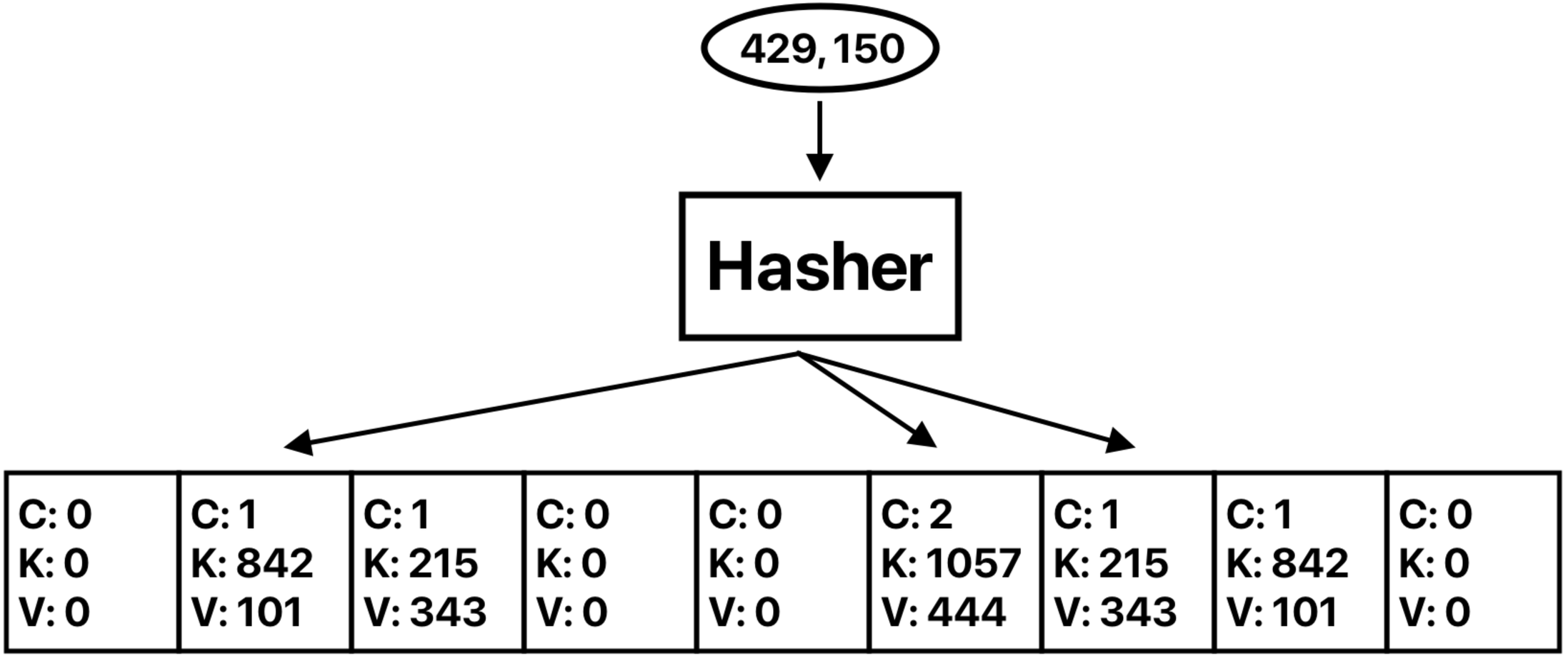
Original IBLT - Insert(x, y)



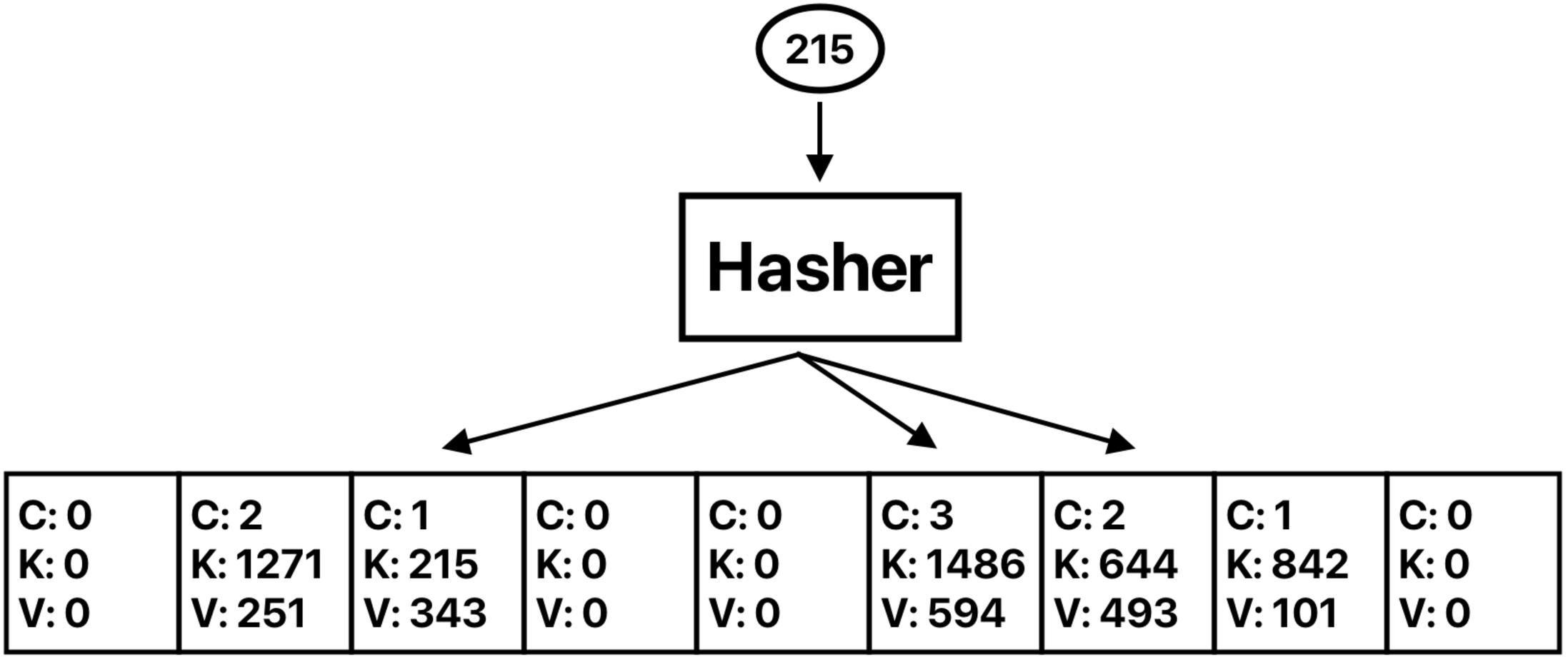
Original IBLT - Delete(x, y)



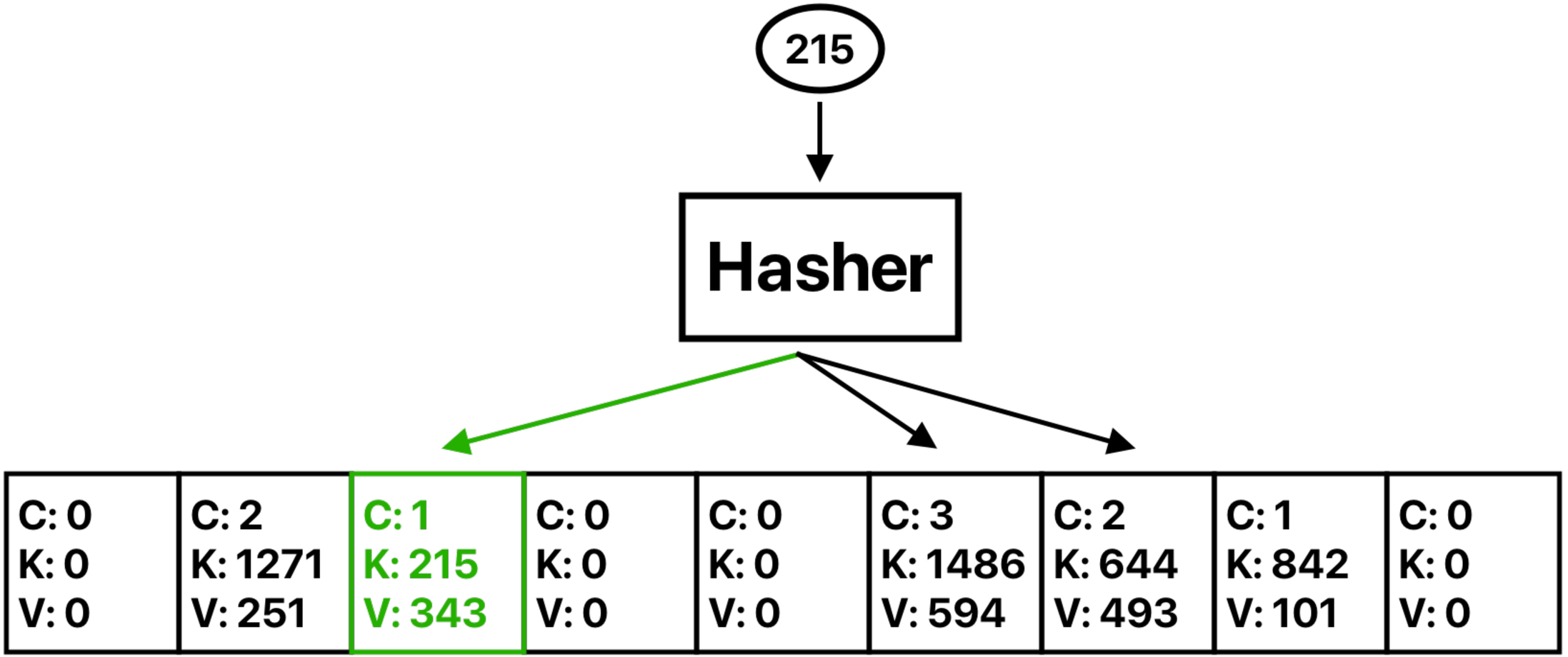
Original IBLT - Delete(x, y)



Original IBLT - Get(x)



Original IBLT - Get(x)



Original IBLT - ListEntries()

```
while exists cell w/ C=1:
  add (key, val) to results
  DELETE(key, val)
```

Result:

C: 0 K: 0 V: 0	C: 2 K: 1271 V: 251	C: 1 K: 215 V: 343	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 3 K: 1486 V: 594	C: 2 K: 644 V: 493	C: 1 K: 842 V: 101	C: 0 K: 0 V: 0

Original IBLT - ListEntries()

```
while exists cell w/ C=1:
  add (key, val) to results
  DELETE(key, val)
```

Result:

C: 0 K: 0 V: 0	C: 2 K: 1271 V: 251	C: 1 K: 215 V: 343	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 3 K: 1486 V: 594	C: 2 K: 644 V: 493	C: 1 K: 842 V: 101	C: 0 K: 0 V: 0

Original IBLT - ListEntries()

```
while exists cell w/ C=1:
  add (key, val) to results
  DELETE(key, val)
```

Result:

(215, 343)

C: 0 K: 0 V: 0	C: 2 K: 1271 V: 251	C: 1 K: 215 V: 343	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 3 K: 1486 V: 594	C: 2 K: 644 V: 493	C: 1 K: 842 V: 101	C: 0 K: 0 V: 0

Original IBLT - ListEntries()

```
while exists cell w/ C=1:
  add (key, val) to results
  DELETE(key, val)
```

Result:

(215, 343)

C: 0 K: 0 V: 0	C: 2 K: 1271 V: 251	C: 1 K: 215 V: 343	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 3 K: 1486 V: 594	C: 2 K: 644 V: 493	C: 1 K: 842 V: 101	C: 0 K: 0 V: 0

Original IBLT - ListEntries()

```
while exists cell w/ C=1:
  add (key, val) to results
  DELETE(key, val)
```

Result:

(215, 343)

C: 0 K: 0 V: 0	C: 2 K: 1271 V: 251	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 2 K: 1271 V: 251	C: 1 K: 429 V: 150	C: 1 K: 842 V: 101	C: 0 K: 0 V: 0

Original IBLT - ListEntries()

```
while exists cell w/ C=1:
  add (key, val) to results
  DELETE(key, val)
```

Result:

(215, 343)

C: 0 K: 0 V: 0	C: 2 K: 1271 V: 251	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 2 K: 1271 V: 251	C: 1 K: 429 V: 150	C: 1 K: 842 V: 101	C: 0 K: 0 V: 0

Original IBLT - ListEntries()

```
while exists cell w/ C=1:
  add (key, val) to results
  DELETE(key, val)
```

Result:

(215, 343)
(842, 101)

C: 0 K: 0 V: 0	C: 2 K: 1271 V: 251	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 2 K: 1271 V: 251	C: 1 K: 429 V: 150	C: 1 K: 842 V: 101	C: 0 K: 0 V: 0

Original IBLT - ListEntries()

```
while exists cell w/ C=1:
  add (key, val) to results
  DELETE(key, val)
```

Result:

(215, 343)
(842, 101)

C: 0 K: 0 V: 0	C: 2 K: 1271 V: 251	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 2 K: 1271 V: 251	C: 1 K: 429 V: 150	C: 1 K: 842 V: 101	C: 0 K: 0 V: 0

Original IBLT - ListEntries()

```
while exists cell w/ C=1:
  add (key, val) to results
  DELETE(key, val)
```

Result:

(215, 343)
(842, 101)

C: 0 K: 0 V: 0	C: 1 K: 429 V: 150	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 1 K: 429 V: 150	C: 1 K: 429 V: 150	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0

Original IBLT - ListEntries()

```
while exists cell w/ C=1:
  add (key, val) to results
  DELETE(key, val)
```

Result:

(215, 343)
(842, 101)

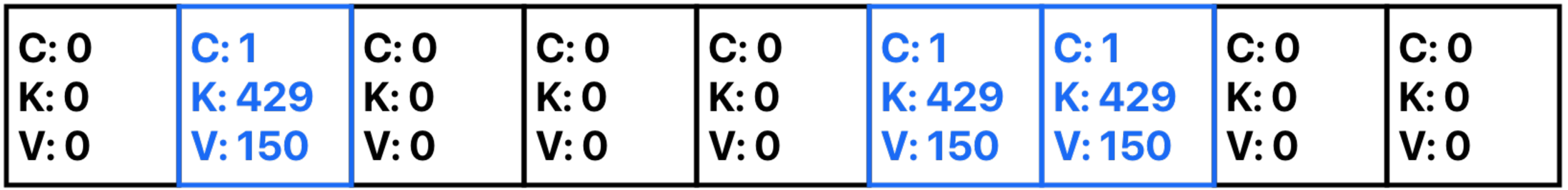
C: 0 K: 0 V: 0	C: 1 K: 429 V: 150	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 1 K: 429 V: 150	C: 1 K: 429 V: 150	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0

Original IBLT - ListEntries()

```
while exists cell w/ C=1:
  add (key, val) to results
  DELETE(key, val)
```

Result:

(215, 343)
(842, 101)
(429, 150)

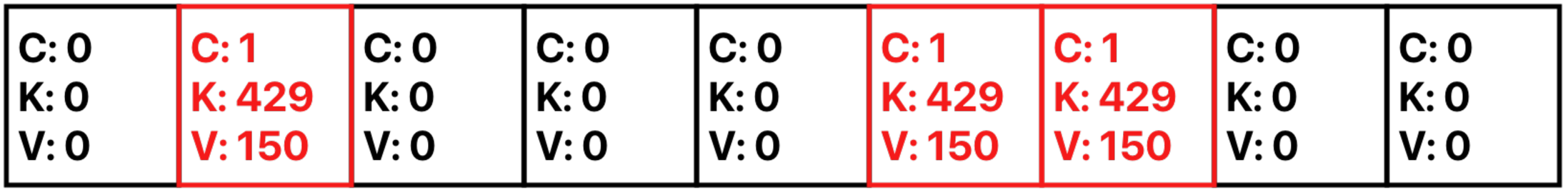


Original IBLT - ListEntries()

```
while exists cell w/ C=1:
  add (key, val) to results
  DELETE(key, val)
```

Result:

(215, 343)
(842, 101)
(429, 150)



Original IBLT - ListEntries()

```
while exists cell w/ C=1:
  add (key, val) to results
  DELETE(key, val)
```

Result:

(215, 343)
(842, 101)
(429, 150)

C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0	C: 0 K: 0 V: 0

✅ Success!

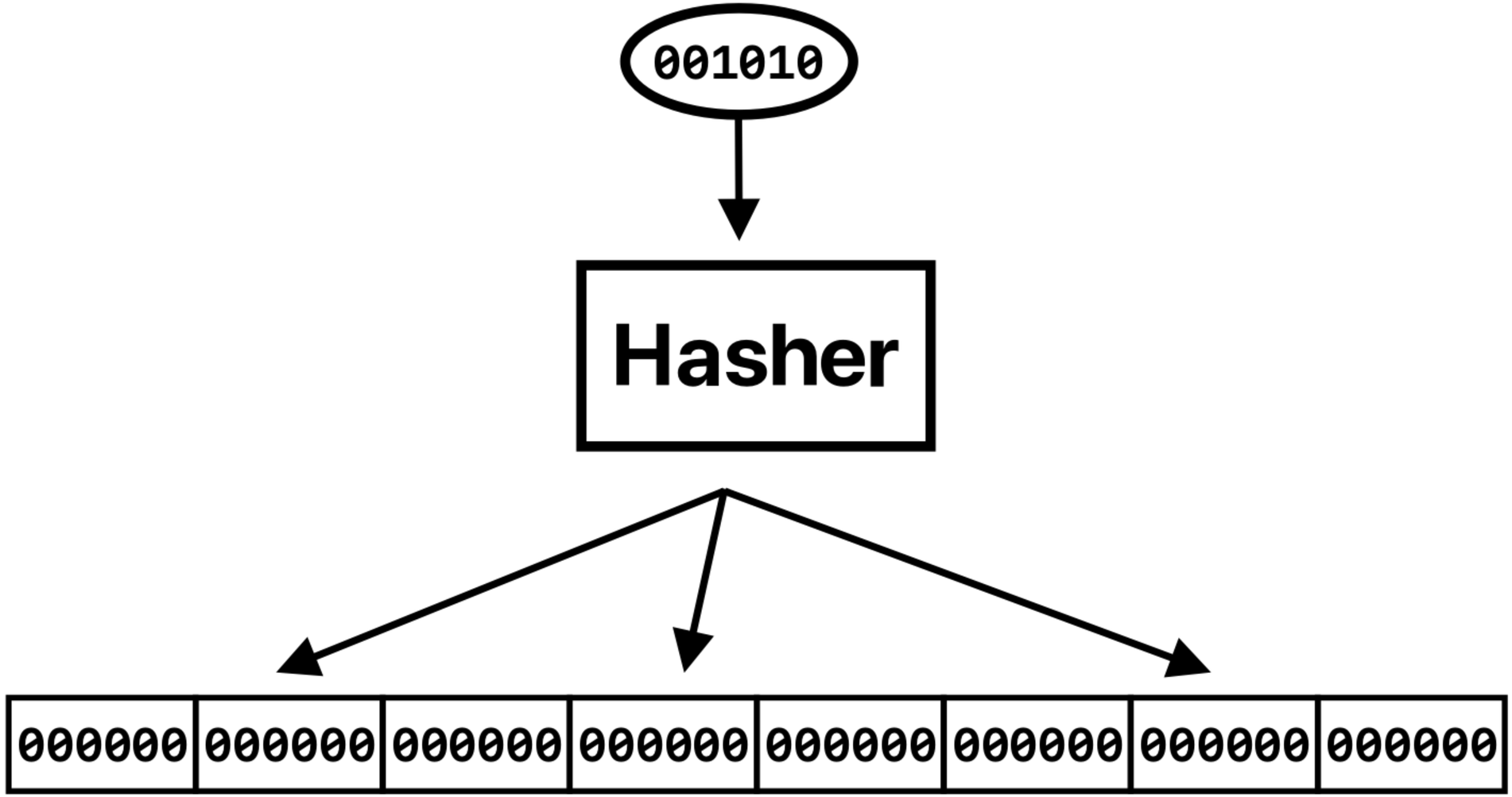
Original IBLT - Limitations

- Extraneous Deletions
- Same Key, Same Values
- Same Key, Different Values

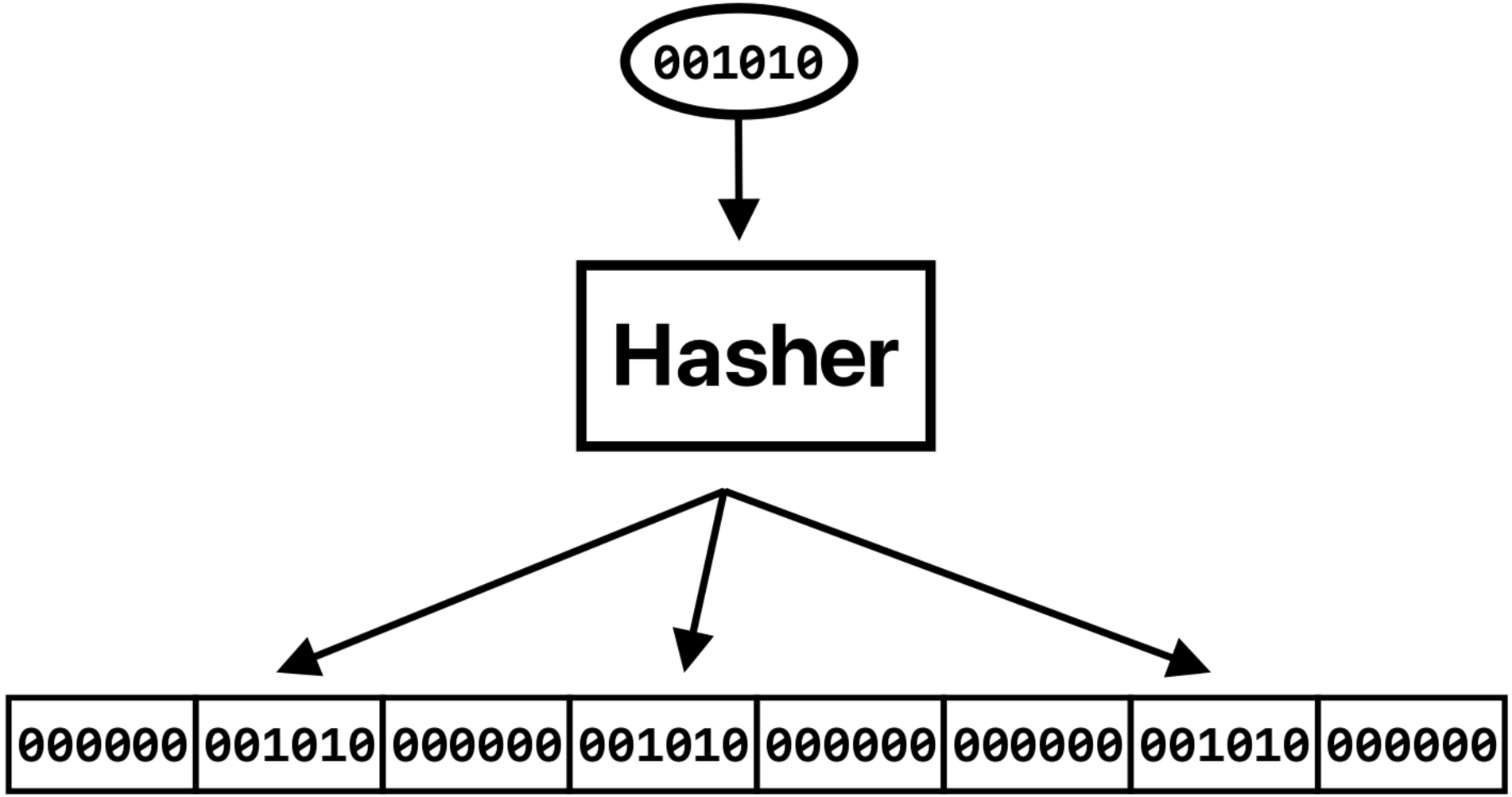
HPW IBLT

- Only stores keys
- Operations:
  - Toggle(x)
  - Merge(other_sketch)
  - ListEntries()

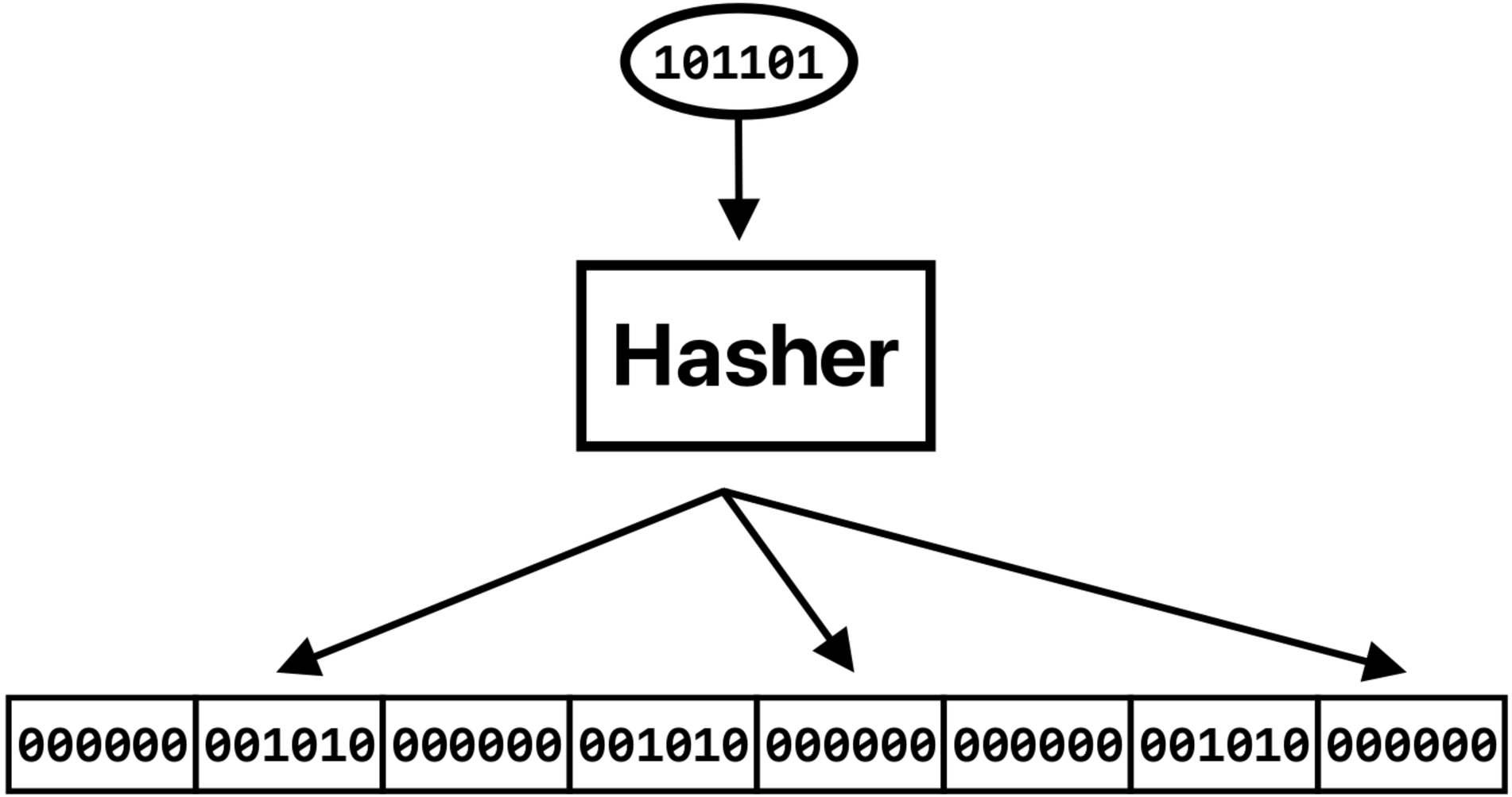
HPW IBLT - Toggle(x)



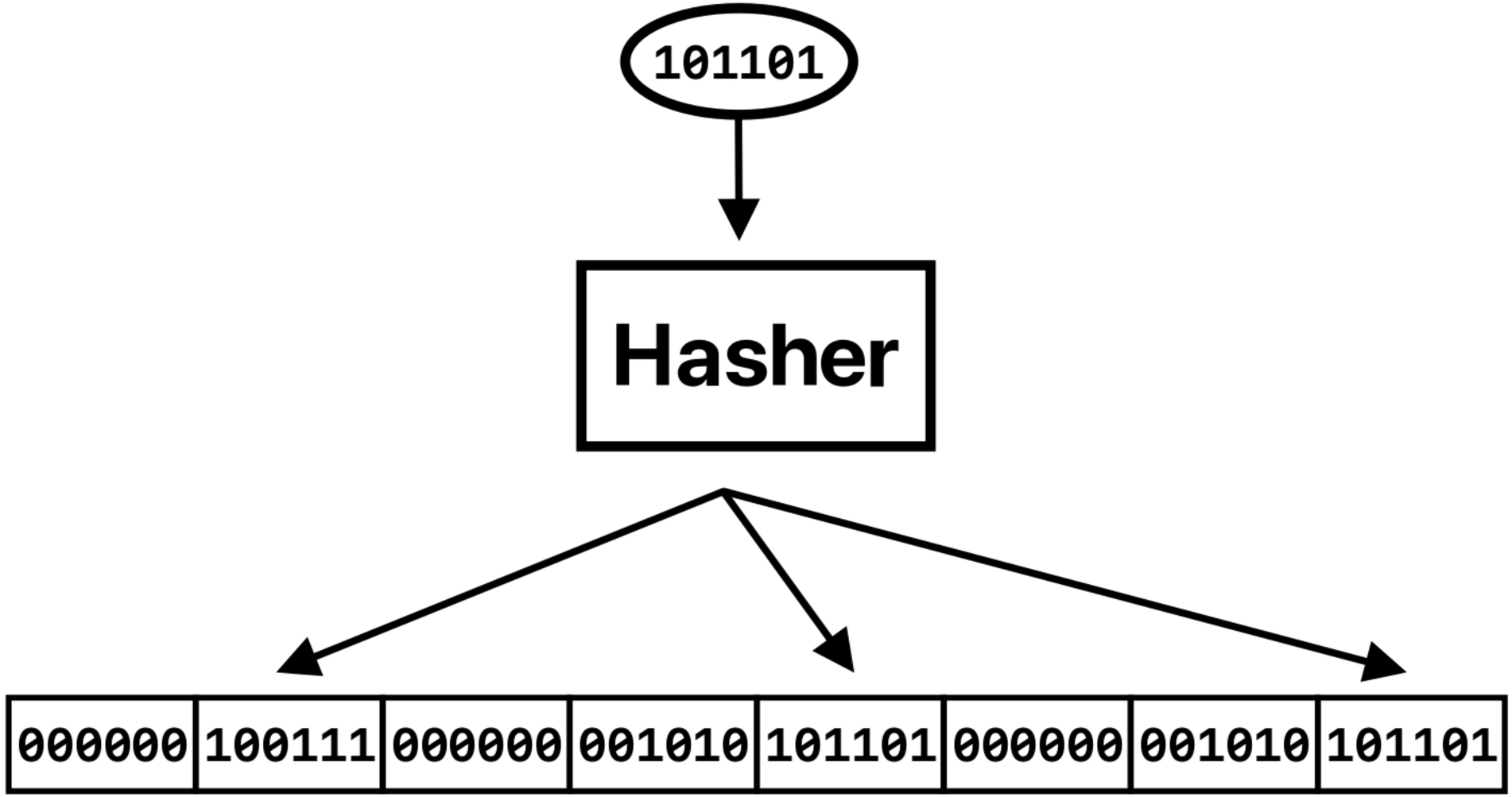
HPW IBLT - Toggle(x)



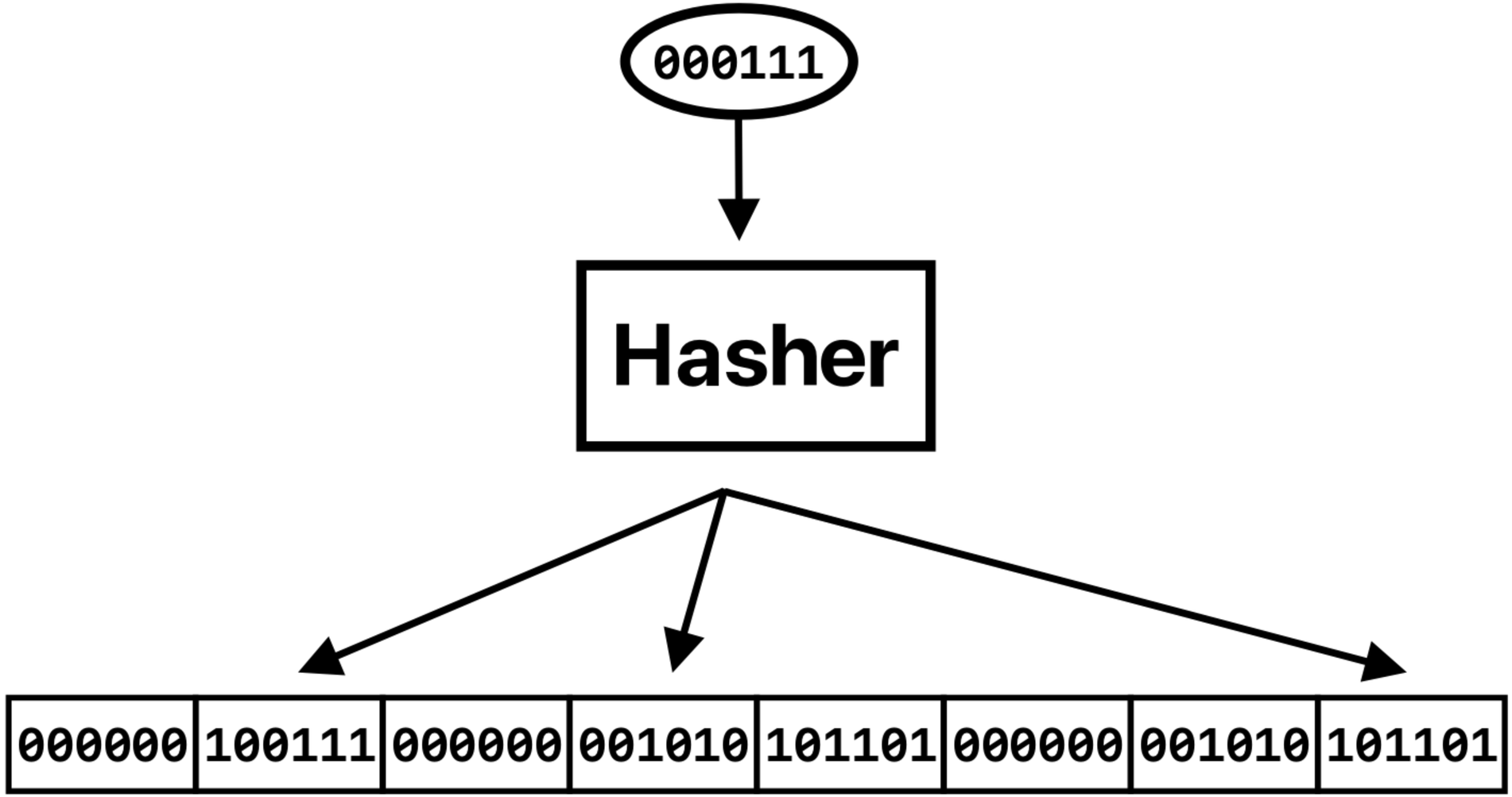
HPW IBLT - Toggle(x)



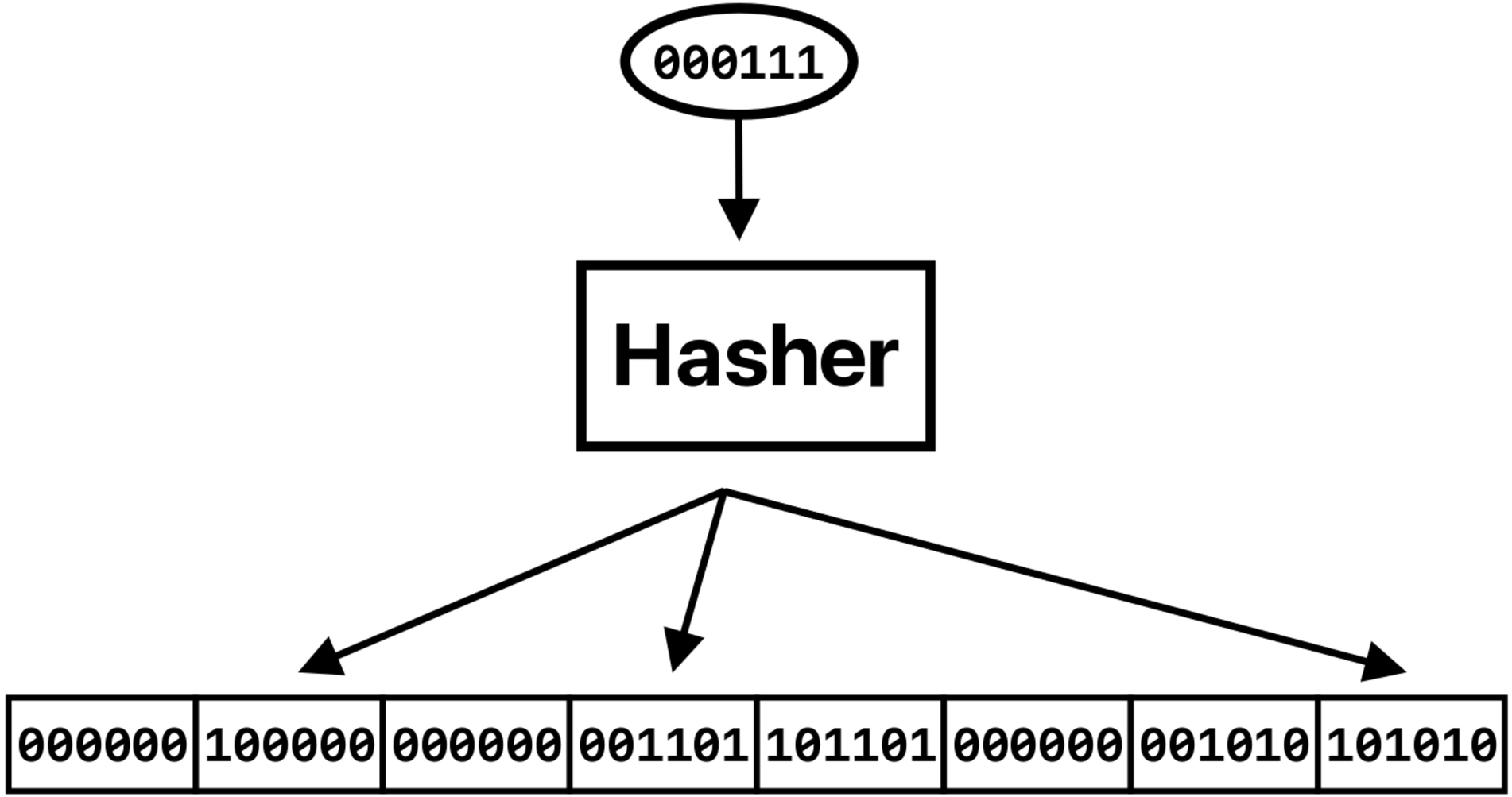
HPW IBLT - Toggle(x)



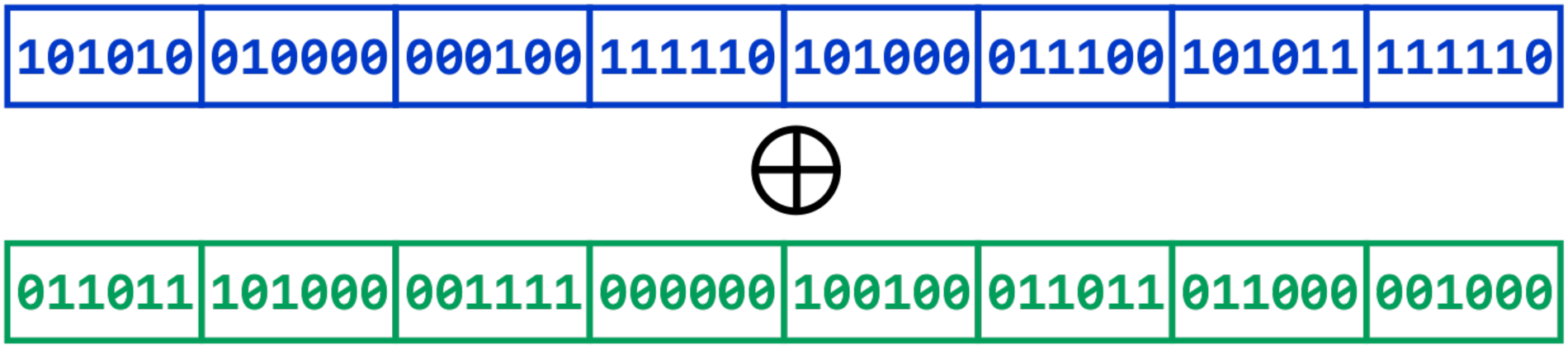
HPW IBLT - Toggle(x)



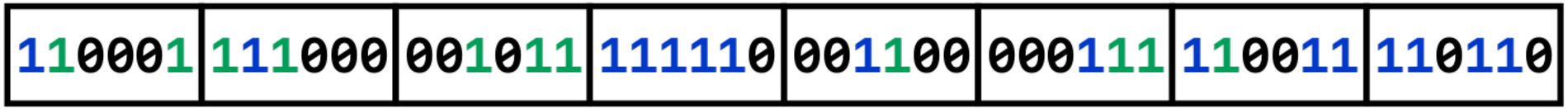
HPW IBLT - Toggle(x)



HPW IBLT - Merge(other_sketch)



HPW IBLT - Merge(other_sketch)



HPW IBLT - ListEntries()

```
while exists pure cell:
  add/remove x to results
  TOGGLE(x)
```

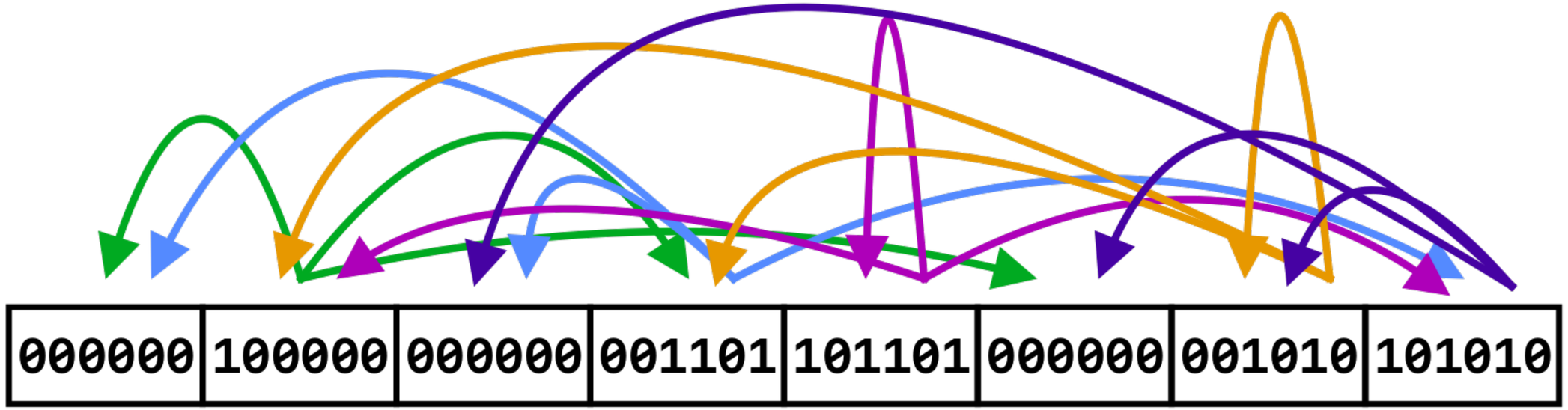
Result:

000000	100000	000000	001101	101101	000000	001010	101010

HPW IBLT - ListEntries()

```
while exists pure cell:
  add/remove x to results
  TOGGLE(x)
```

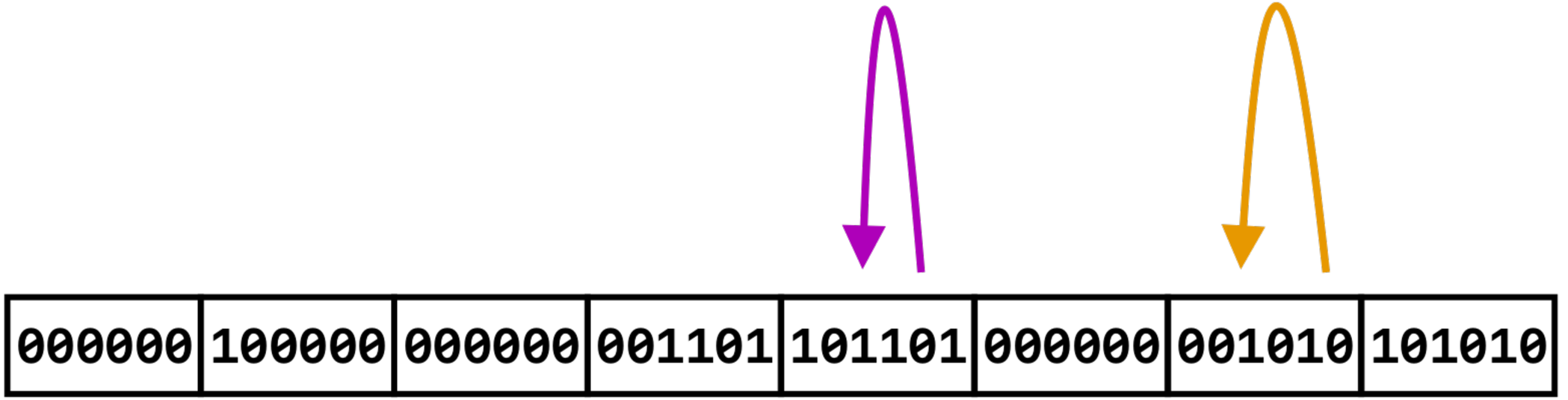
Result:



HPW IBLT - ListEntries()

```
while exists pure cell:
  add/remove x to results
  TOGGLE(x)
```

Result:

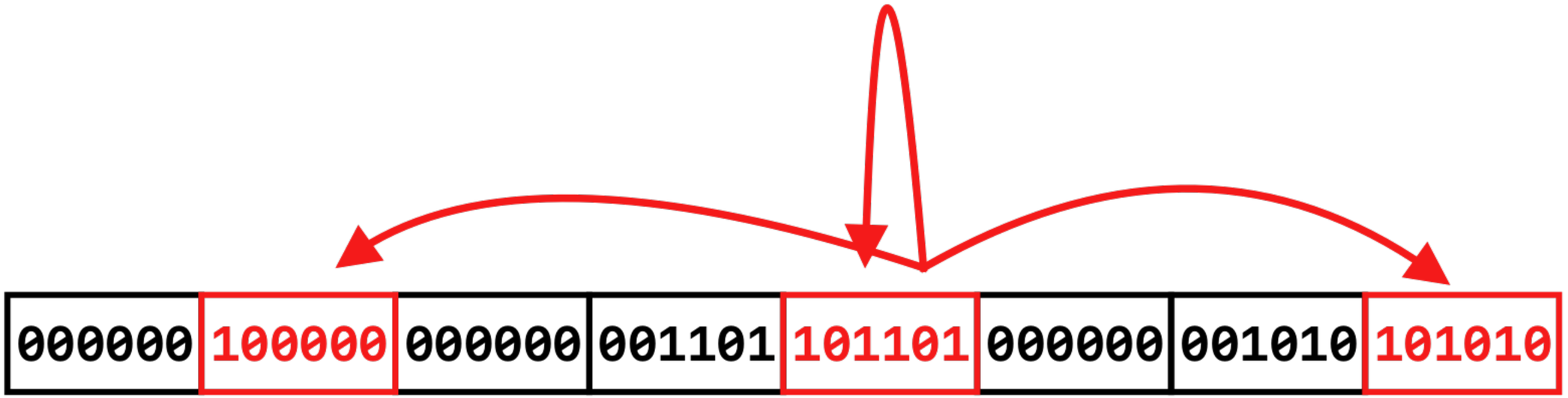


HPW IBLT - ListEntries()

```
while exists pure cell:
  add/remove x to results
  TOGGLE(x)
```

Result:

101101

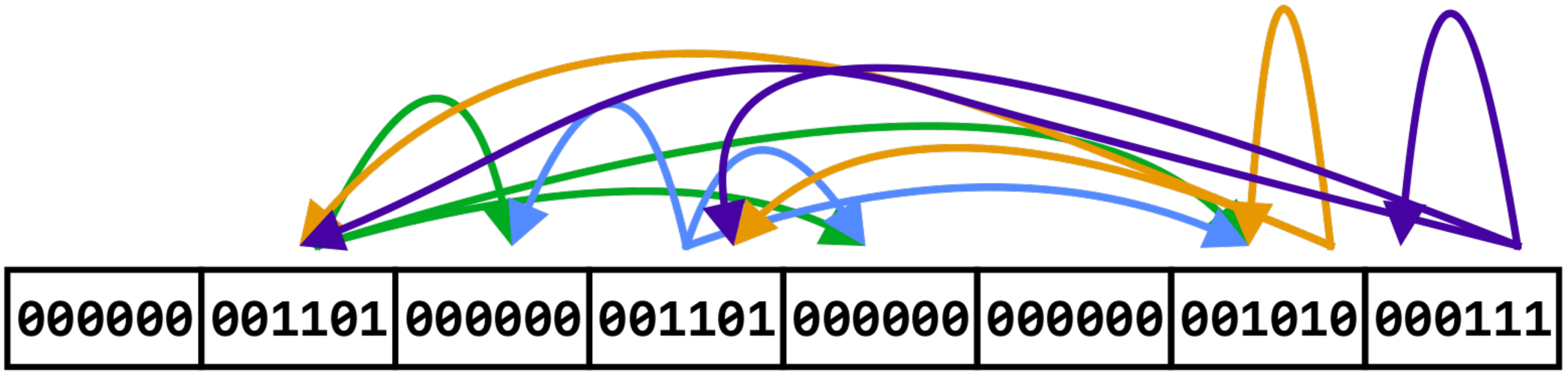


HPW IBLT - ListEntries()

```
while exists pure cell:
  add/remove x to results
  TOGGLE(x)
```

Result:

101101

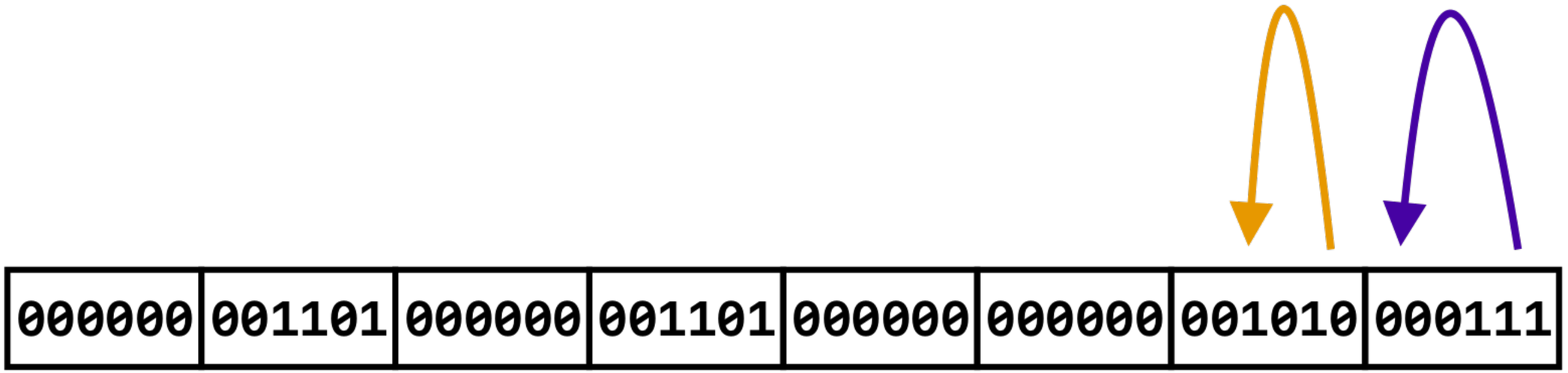


HPW IBLT - ListEntries()

```
while exists pure cell:
  add/remove x to results
  TOGGLE(x)
```

Result:

101101

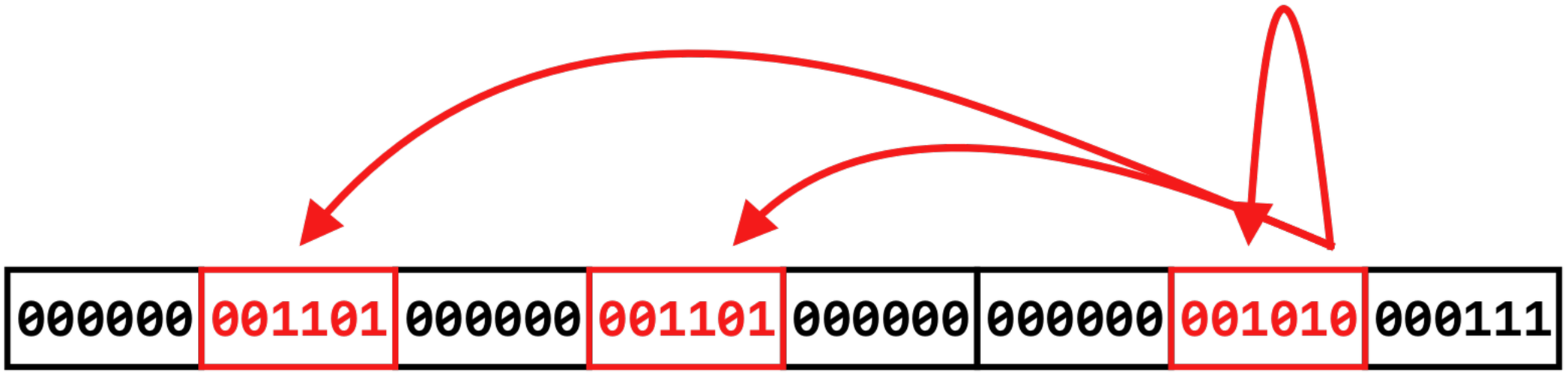


HPW IBLT - ListEntries()

```
while exists pure cell:
  add/remove x to results
  TOGGLE(x)
```

Result:

101101
001010

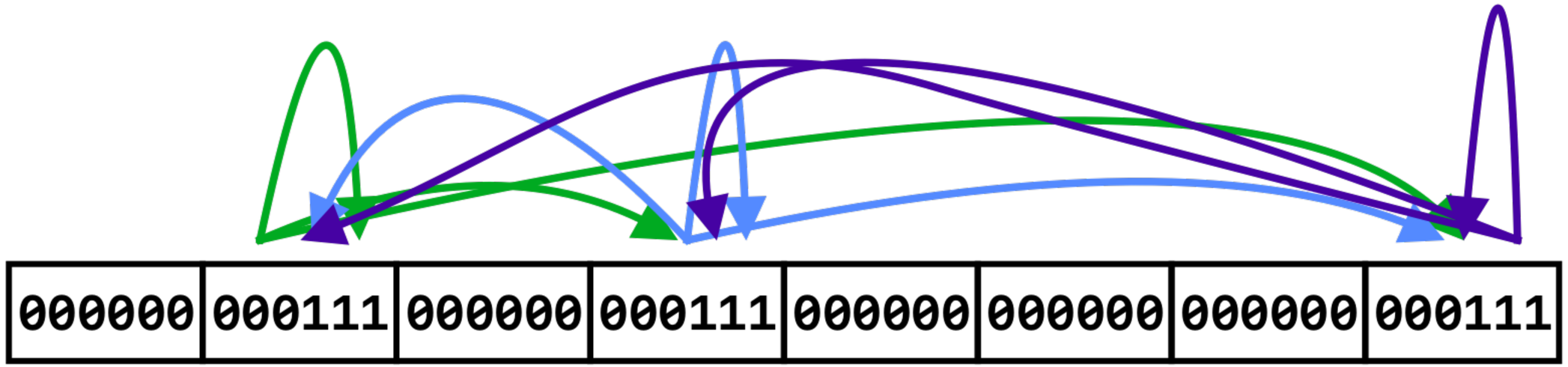


HPW IBLT - ListEntries()

```
while exists pure cell:
  add/remove x to results
  TOGGLE(x)
```

Result:

101101
001010

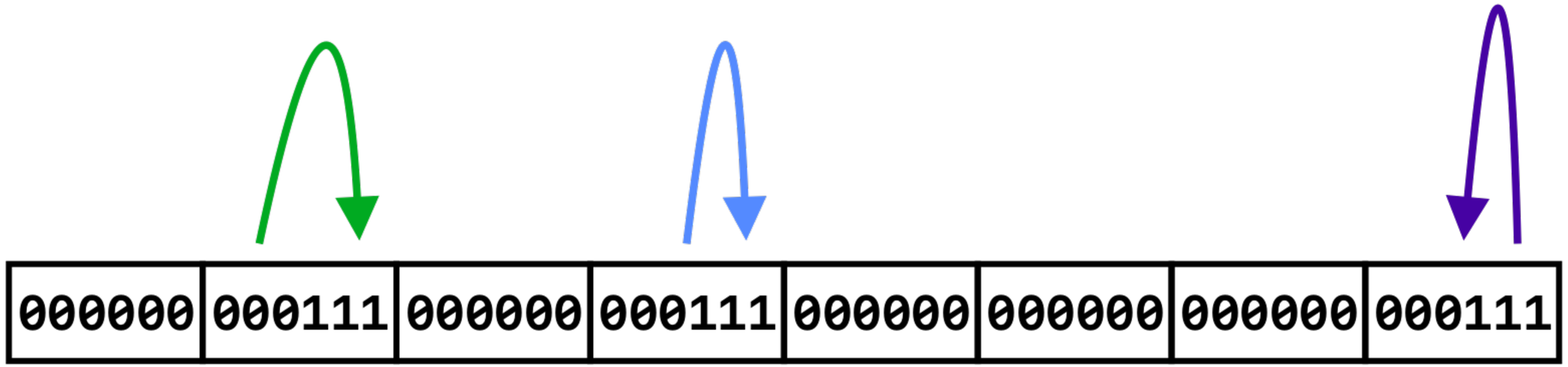


HPW IBLT - ListEntries()

```
while exists pure cell:
  add/remove x to results
  TOGGLE(x)
```

Result:

101101
001010

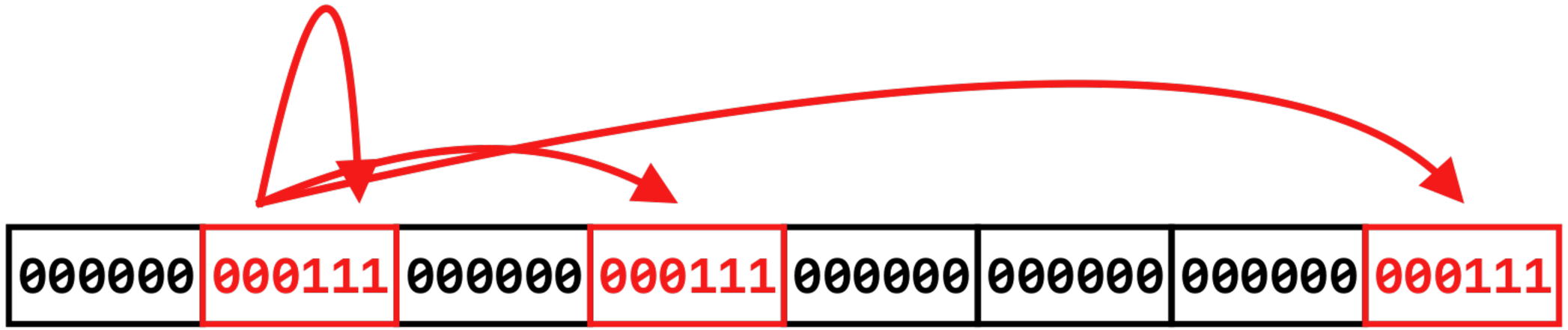


HPW IBLT - ListEntries()

```
while exists pure cell:
  add/remove x to results
  TOGGLE(x)
```

Result:

101101
001010
000111



HPW IBLT - ListEntries()

```
while exists pure cell:
  add/remove x to results
  TOGGLE(x)
```

Result:

101101
001010
000111

000000	000000	000000	000000	000000	000000	000000	000000

✅ **Success!**

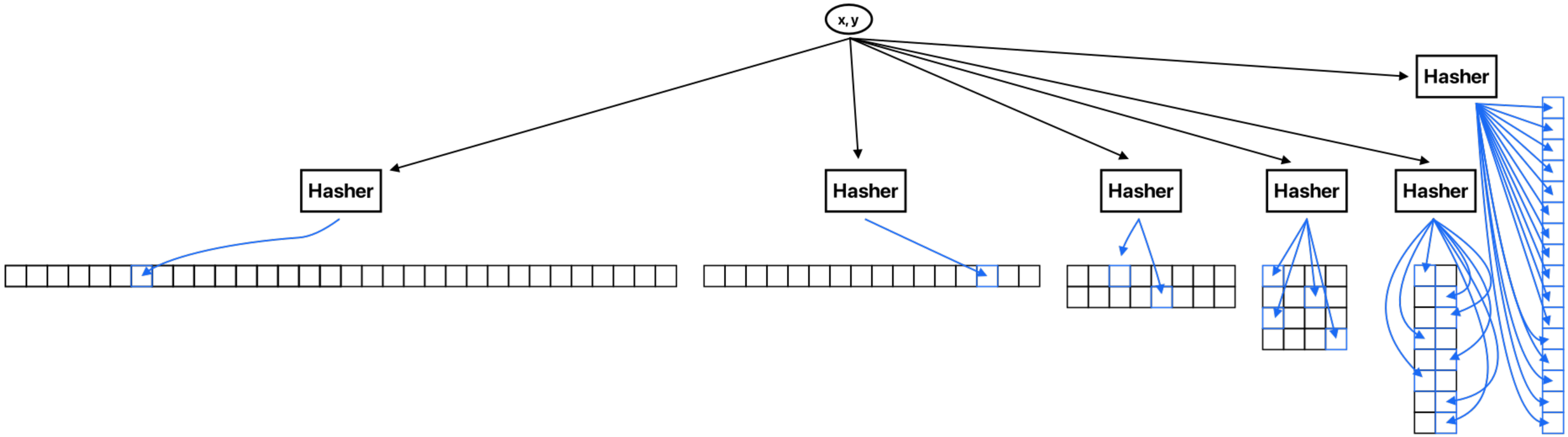
HPW IBLT Quirks

- Only stores keys
- Anomalies are mostly self-correcting
- No partial failure, all or nothing.

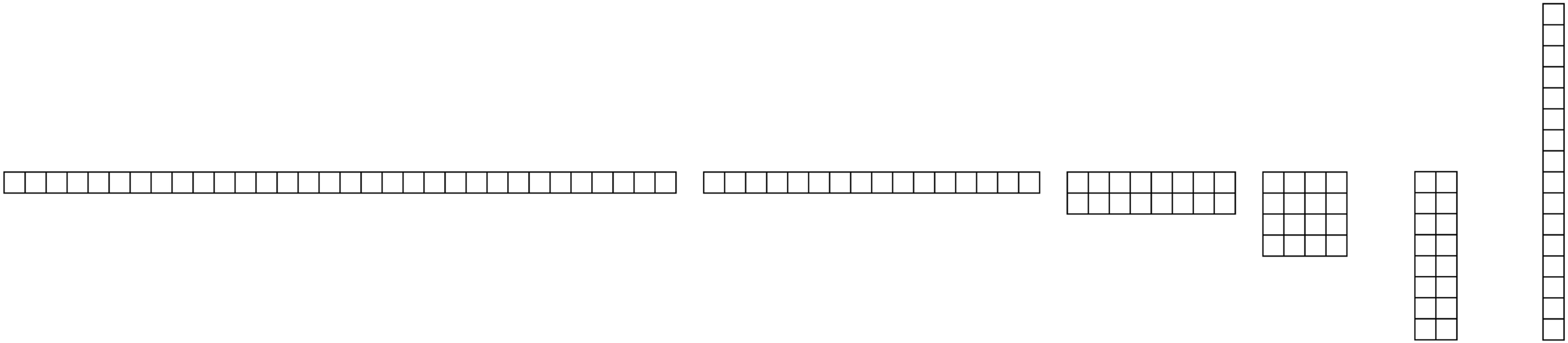
Stacked IBLT

- Key/Value Store
- Uses multiple "Basic IBLTs"
- Operations:
  - Insert(key, value)
  - Remove(key, value)
  - ListEntries()

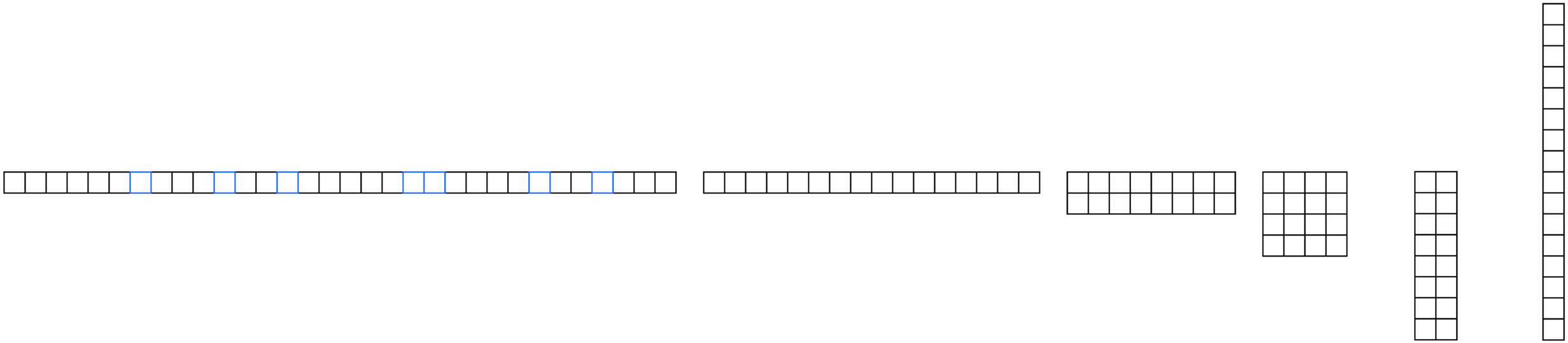
Stacked IBLT



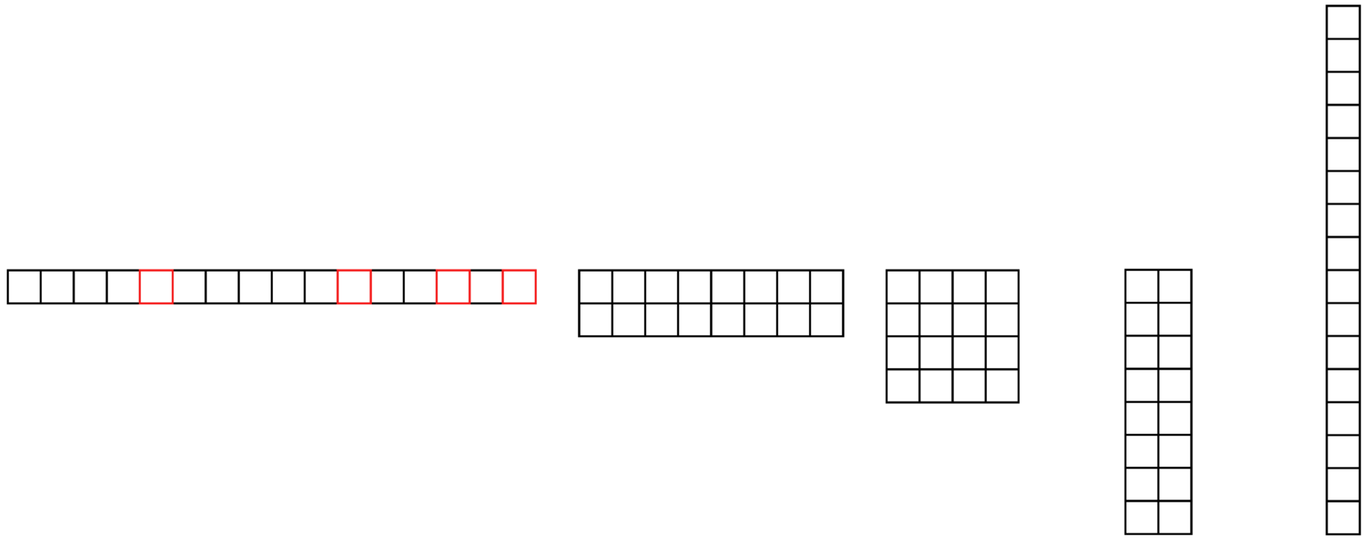
Stacked IBLT - ListEntries()



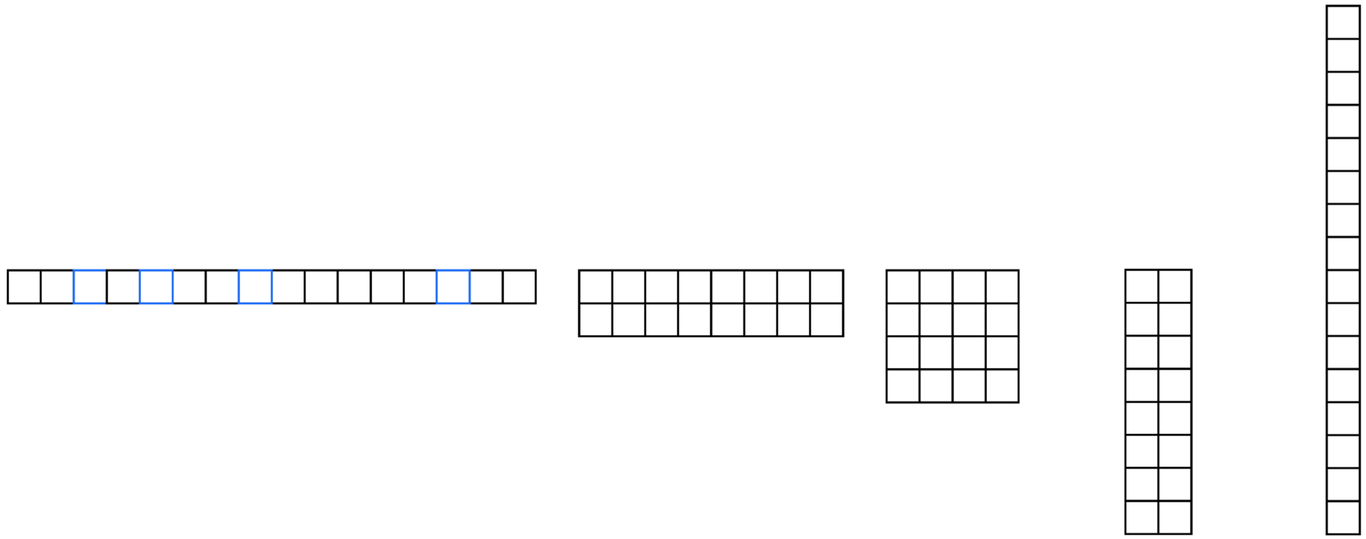
Stacked IBLT - ListEntries()



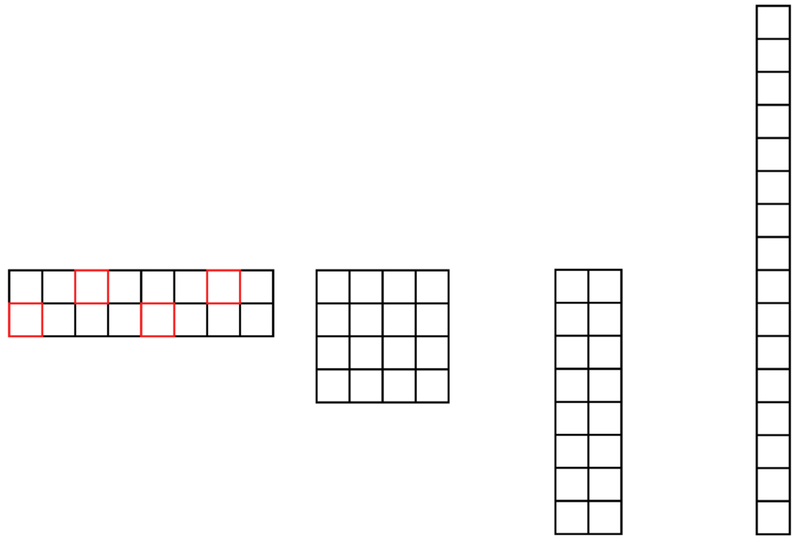
Stacked IBLT - ListEntries()



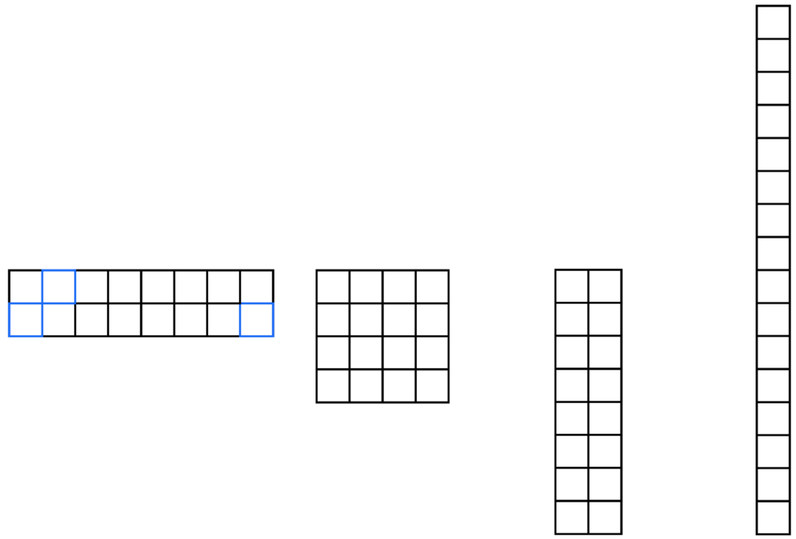
Stacked IBLT - ListEntries()



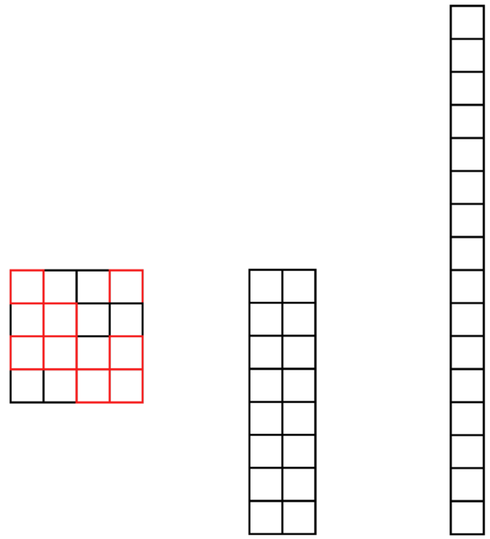
Stacked IBLT - ListEntries()



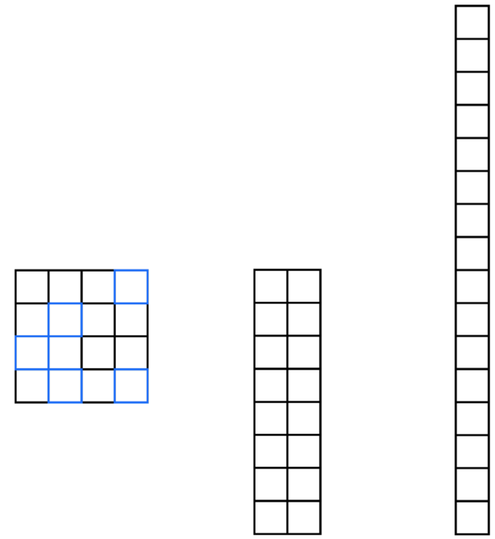
Stacked IBLT - ListEntries()



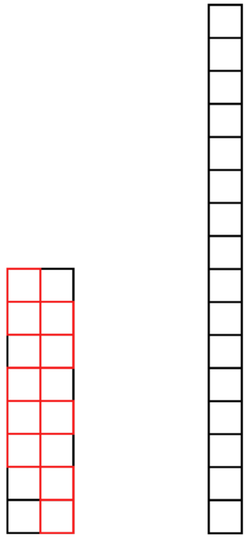
Stacked IBLT - ListEntries()



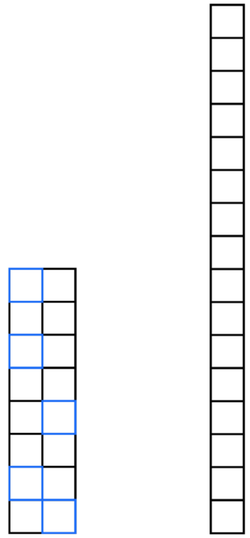
Stacked IBLT - ListEntries()



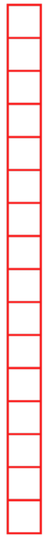
Stacked IBLT - ListEntries()



Stacked IBLT - ListEntries()



Stacked IBLT - ListEntries()



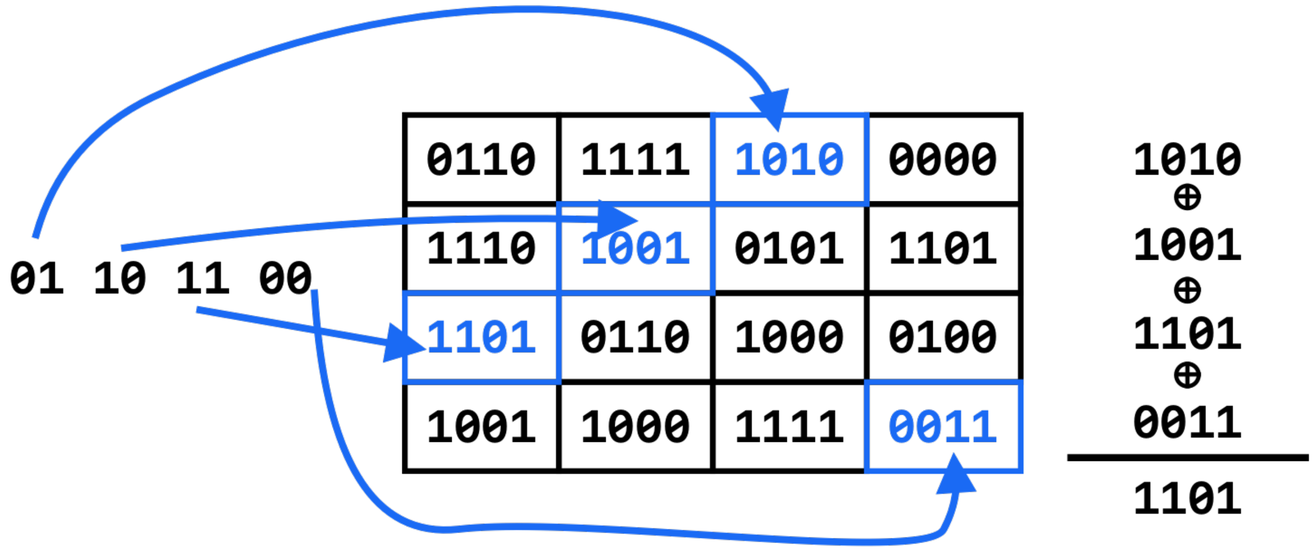
Stacked IBLT - ListEntries()

Stacked IBLT Quirks

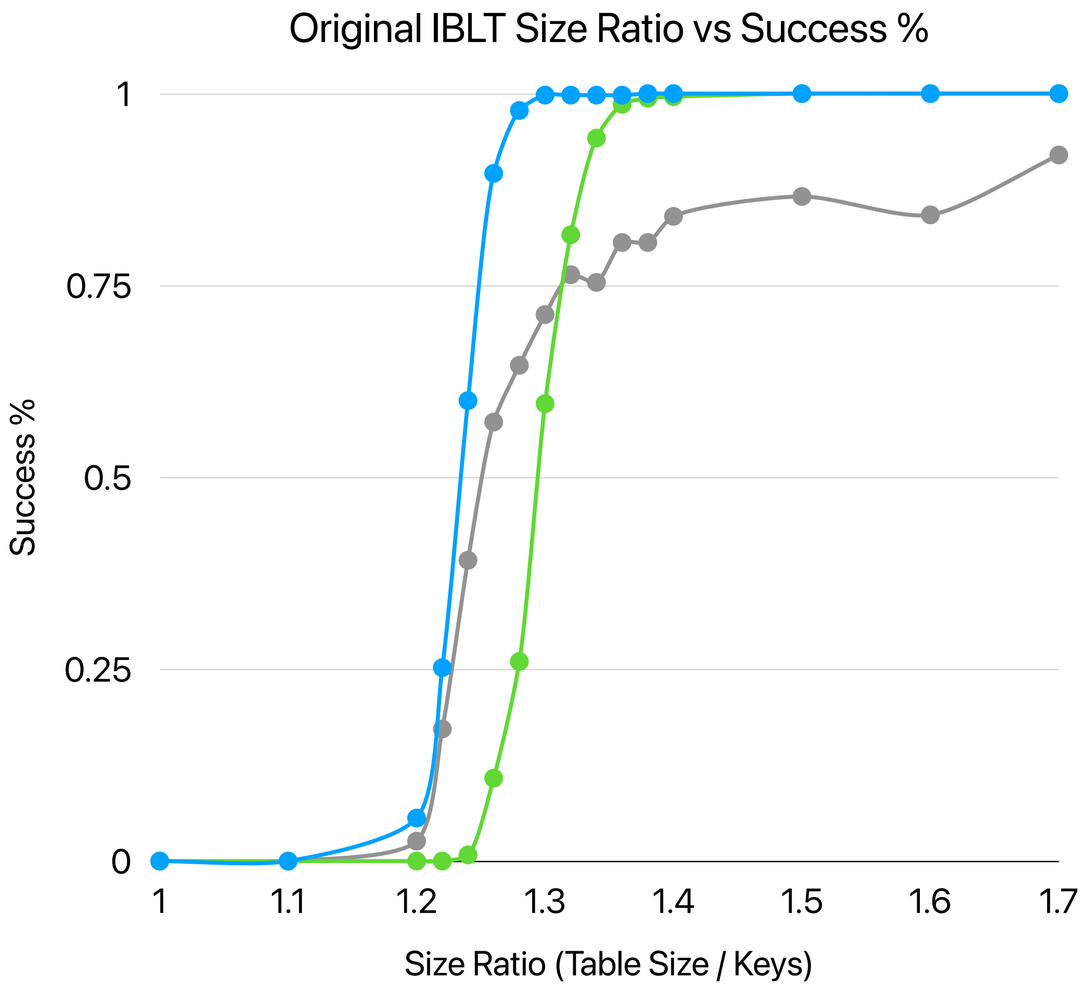
- No individual key->value access
- Supports partial decodes, like Original
- Comes with more configuration options

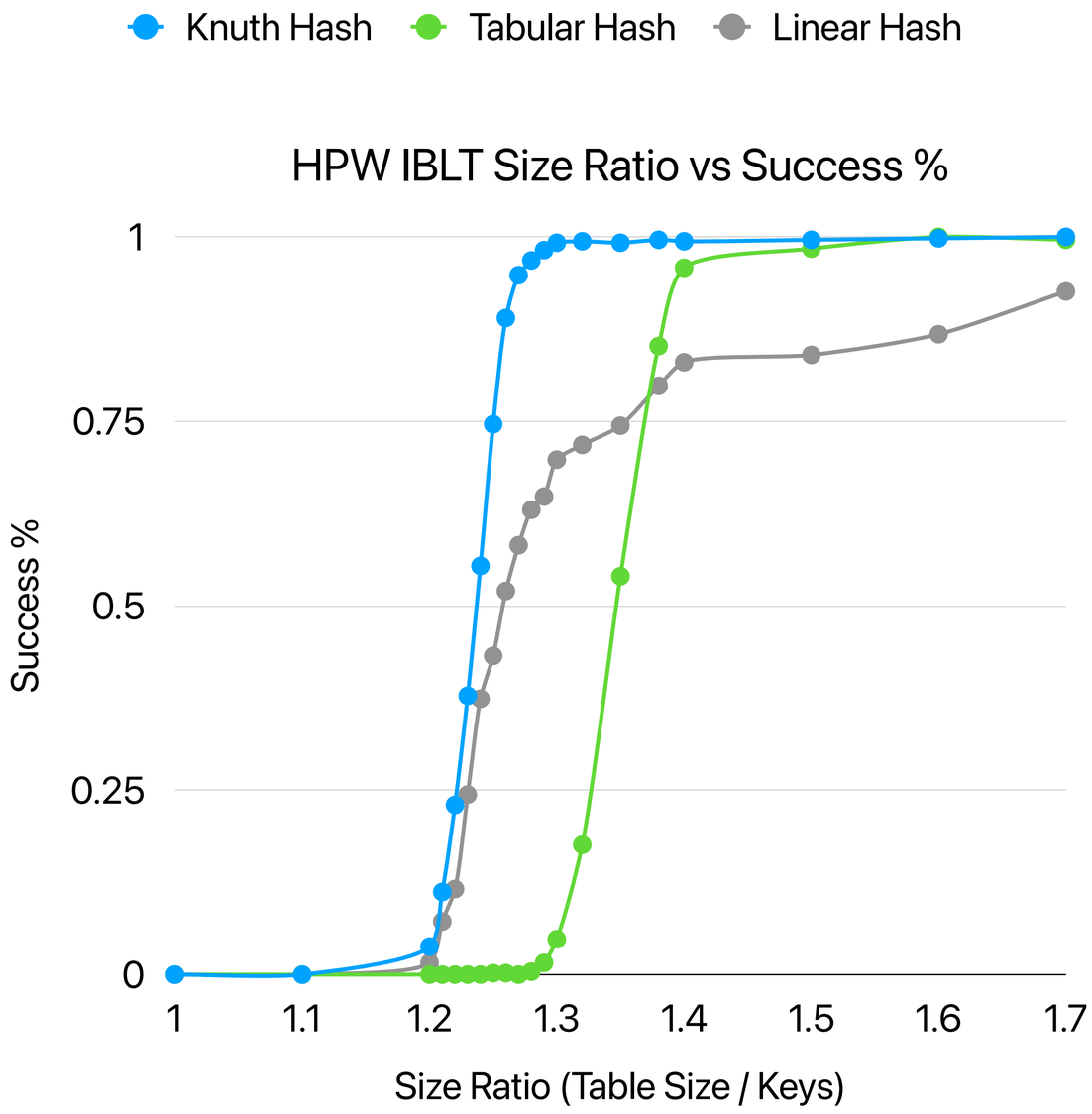
Hash Functions

- Linear Hash:
  - `ax+b % p % array_size`
- Knuth's Multiplicative Hash:
  - `ax >> (w-b) % array_size`
- Tabular Hash



Original & HPW IBLT





500 trials per dot - k=3 hash functions

Stacked IBLT Parameters

- C: relates to column size in Basic IBLTs
- C_0 and δ: controls # of single row Basic IBLTs & column count of multirow Basic IBLTs
  - C_0 is hardcoded
  - δ is set by user
  - $\tau = C^*\log_2(1/\delta)$

$$
\begin{array}{l}
\textsf{Init}(\boldsymbol{h}) \\
\hline
\textbf{for } 0 \le i < \lg(n) - \lg(\tau) \\
\quad T_i := \textsf{BasicInit}(1, \lceil Cn2^{-i} \rceil, \boldsymbol{h}_i) \\
\textbf{for } 0 \le i < \lg(\tau) \\
\quad i' := \lfloor \lg(n) - \lg(\tau) \rfloor + i \\
\quad T_{i'} := \textsf{BasicInit}(2^i, \lceil C\tau 2^{-i} \rceil, \boldsymbol{h}_{i'}) \\
\textbf{return } (T_0, \ldots, T_{\lceil \lg n \rceil - 1})
\end{array}
$$

```
double tau = C_0 * log2(1/error_rate);
int logtau = std::ceil(log2(tau));
int singlerow_count = sketchCount - logtau;
```

```
// Single Row Columns
for (int index = 0; index < singlerow_count; index++) {
    int columns = std::ceil(C * threshold * std::pow(2, -index));
    this->addNextSketch(1, columns);
}

// Multirow Columns
for (int index = 0; index < logtau; index++) {
    int columns = std::ceil(C * tau * std::pow(2, -index));
    int rows = std::pow(2, index);

    this->addNextSketch(rows, columns);
}
```

Stacked Results

- C is the most responsive
  - Any C > 5/e is robust
- τ ($C_0 * \log_2(1/\delta)$) is less responsive and less correlated at times

Stacked Success: C vs C_0 (δ=0.1)

C	5	10	20
8/e		1	1
6/e		1	
5/e	1	1	
4.5/e		1	
4.25/e	0.99	0.98	
4.1/e	0.89	0.91	
4.05/e	0.85	0.948	
4/e		0.81	0.65

Summary

- Stacked IBLT is complex to use
  - Practical benefits over Original aren't substantial
- KnuthHash performs best
  - Original and HPW perform well with KnuthHash & Space Ratio >= 1.3
- Future areas to explore:
  - Store data types other than ints
  - Handle more anomalies
  - Performance / Memory Usage

Sources

- Fleischhacker, N., Larsen, K. G., Obremski, M., & Simkin, M. (2024). Invertible bloom lookup tables with less memory and randomness. 32nd Annual European Symposium on Algorithms (ESA 2024), 54:1-54:17. https://doi.org/10.4230/LIPIcs.ESA.2024.54
- Goodrich, M. T., & Mitzenmacher, M. (2011). Invertible bloom lookup tables. 2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton), 792–799. https://doi.org/10.1109/Allerton.2011.6120248
- Houen, J. B. T., Pagh, R., & Walzer, S. (2023). Simple set sketching. Symposium on Simplicity in Algorithms. https://doi.org/10.1137/1.9781611977585

lkellar/iblt