Group-Key Indices Lookup and Maintenance

Martin Faust, David Schwalb, Jens Krueger
and Hasso Plattner

Hasso Plattner Institute, Potsdam, Germany
at ADMS 2012

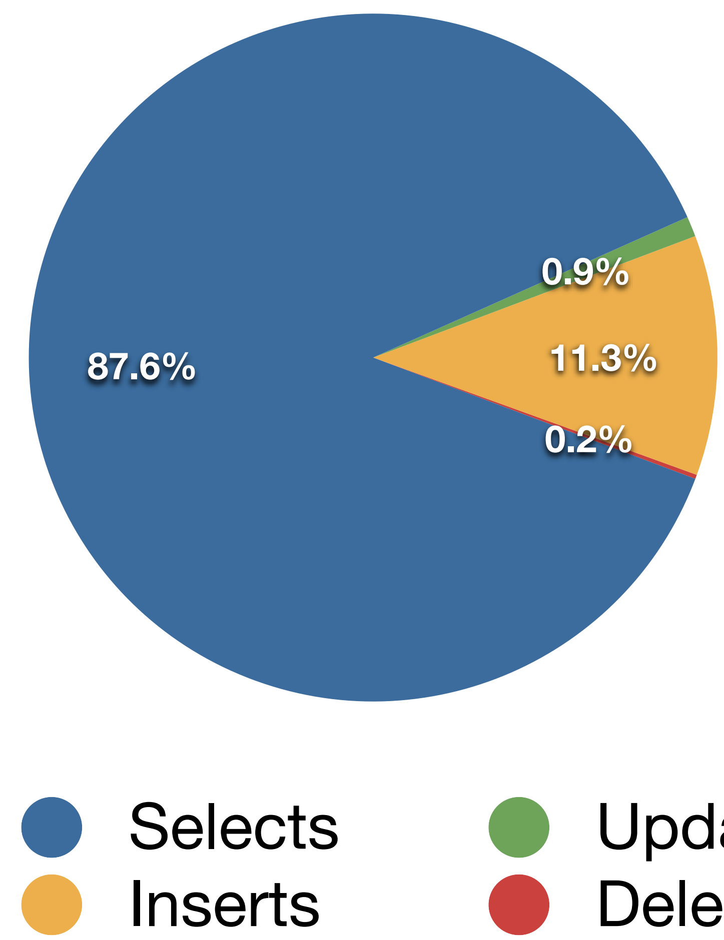
Agenda

- Background
  - Motivation
  - Dictionary encoded In-Memory Column Store
- Group-Key Indices
  - Structure & Lookup
  - Creation from Scratch
  - Integration of Maintenance into the Merge Process
- Performance Results

Motivation

- Indices to speed up query processing of selections
- Memory traffic to assess lookup and maintenance costs
- Integration with merge process to minimize maintenance costs

Query distribution by elapsed time (Customer System)

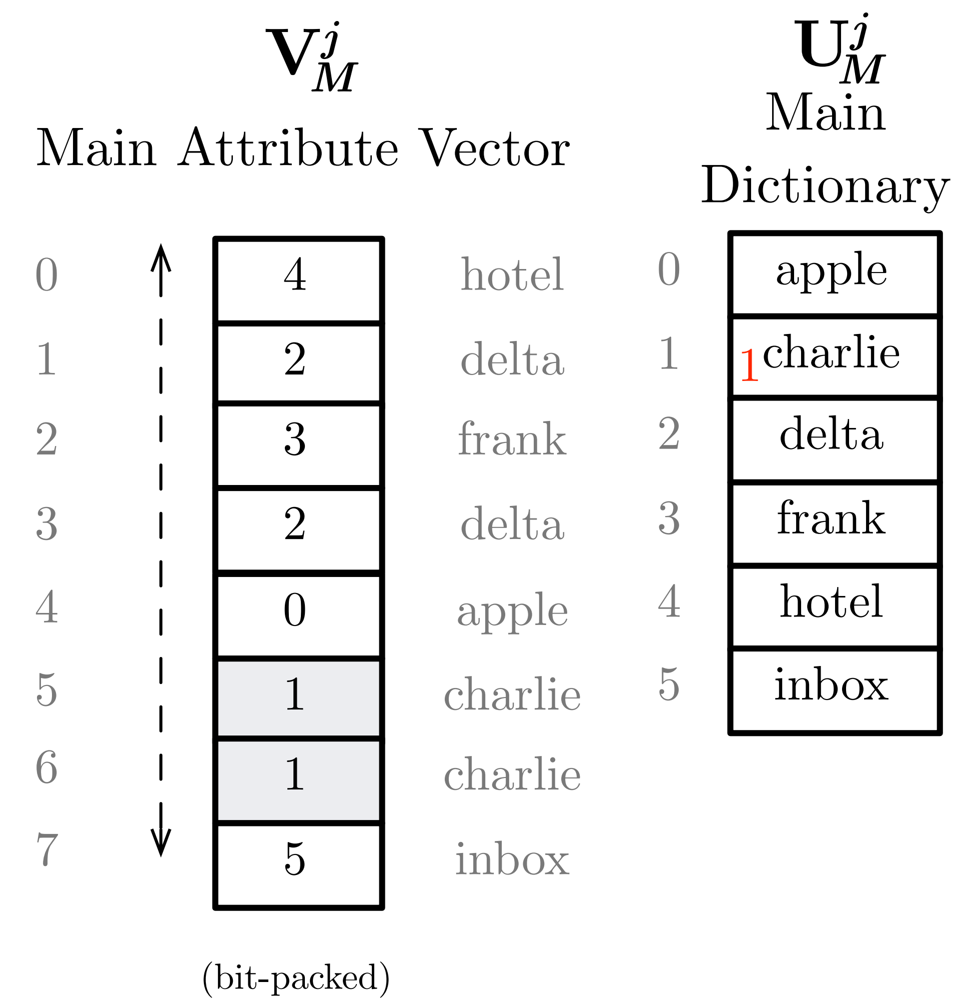


In-Memory Column Store

- Column consists of typically large, read-only main partition and smaller, writable delta partition
- Insert-only mode
- Queries are executed on main and delta partition simultaneously

Main Partition

- The value domain is encoded in a **sorted dictionary**
- Column is stored as vector of bit-packed value-ids, called the **attribute vector**
- High compression and high scan speed



Delta Partition

- Delta partition's values are stored **uncompressed** in a vector (**D**)
- **CSB+ tree index** stores value to position information (**T**)
- Fast operations on small delta partition

Merge Process

- Delta is uncompressed and therefore large
- The performance decreases, if the delta partition grows too large
- Merge main and delta partition from time to time to create a new main partition

Group-Key Index Structure I

- Inverted index
- Index is only applied to the main partition
- Index maps value-ids to positions
- Index consists of two bit-packed structures: index offsets (**I**) and index postings (**P**)

Group-Key Index Structure II

$\mathbf{V}_M^j$
Main Attribute Vector

0	4	hotel
1	2	delta
2	3	frank
3	2	delta
4	0	apple
5	1	charlie
6	1	charlie
7	5	inbox

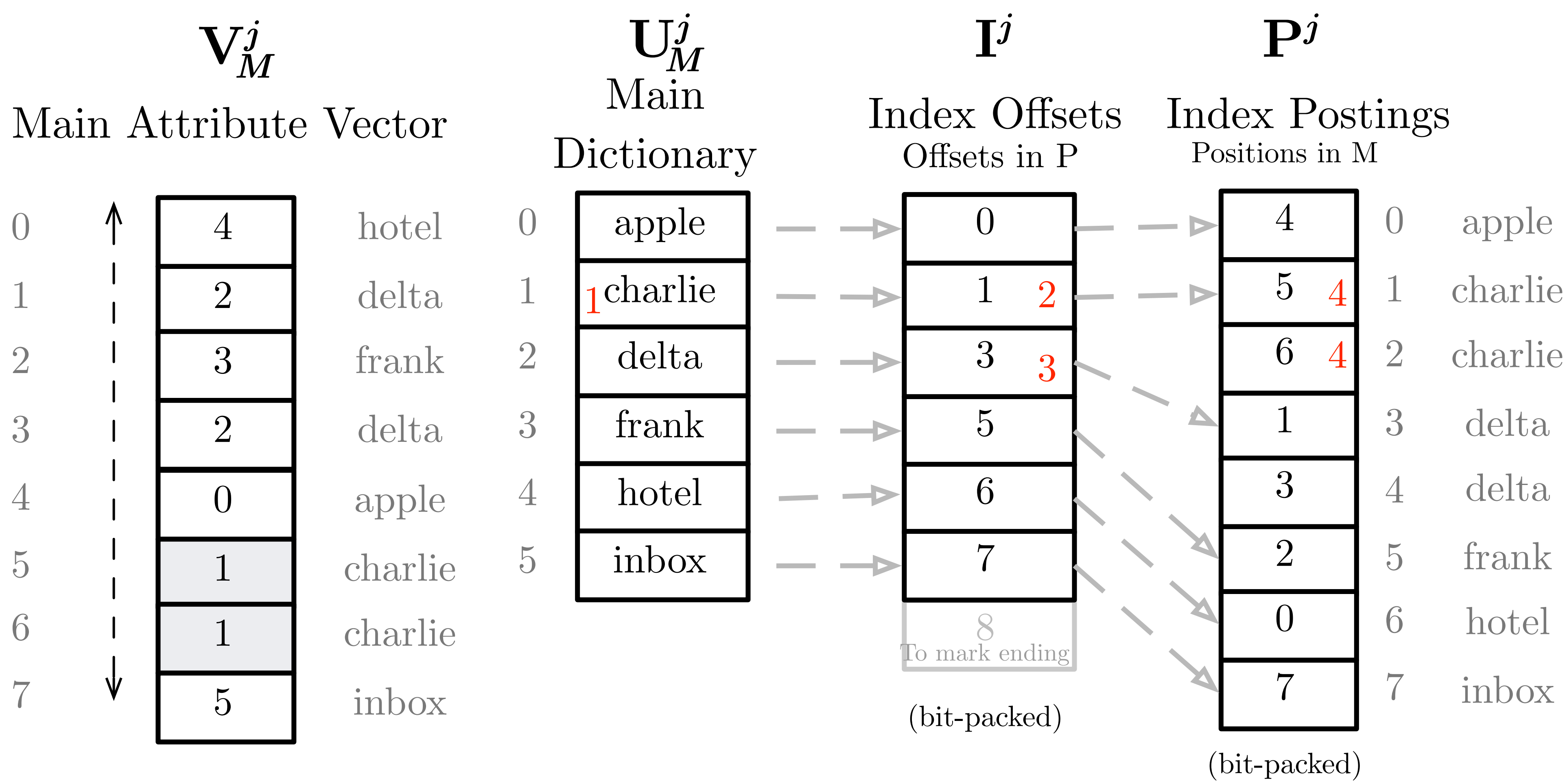
(bit-packed)

$\mathbf{U}_M^j$
Main Dictionary

0	apple
1	1 charlie
2	delta
3	frank
4	hotel
5	inbox

$$\mathbf{E}_C^j = \lceil \log_2 |\mathbf{U}_\mathbf{M}^j| \rceil$$

Group-Key Index Structure II



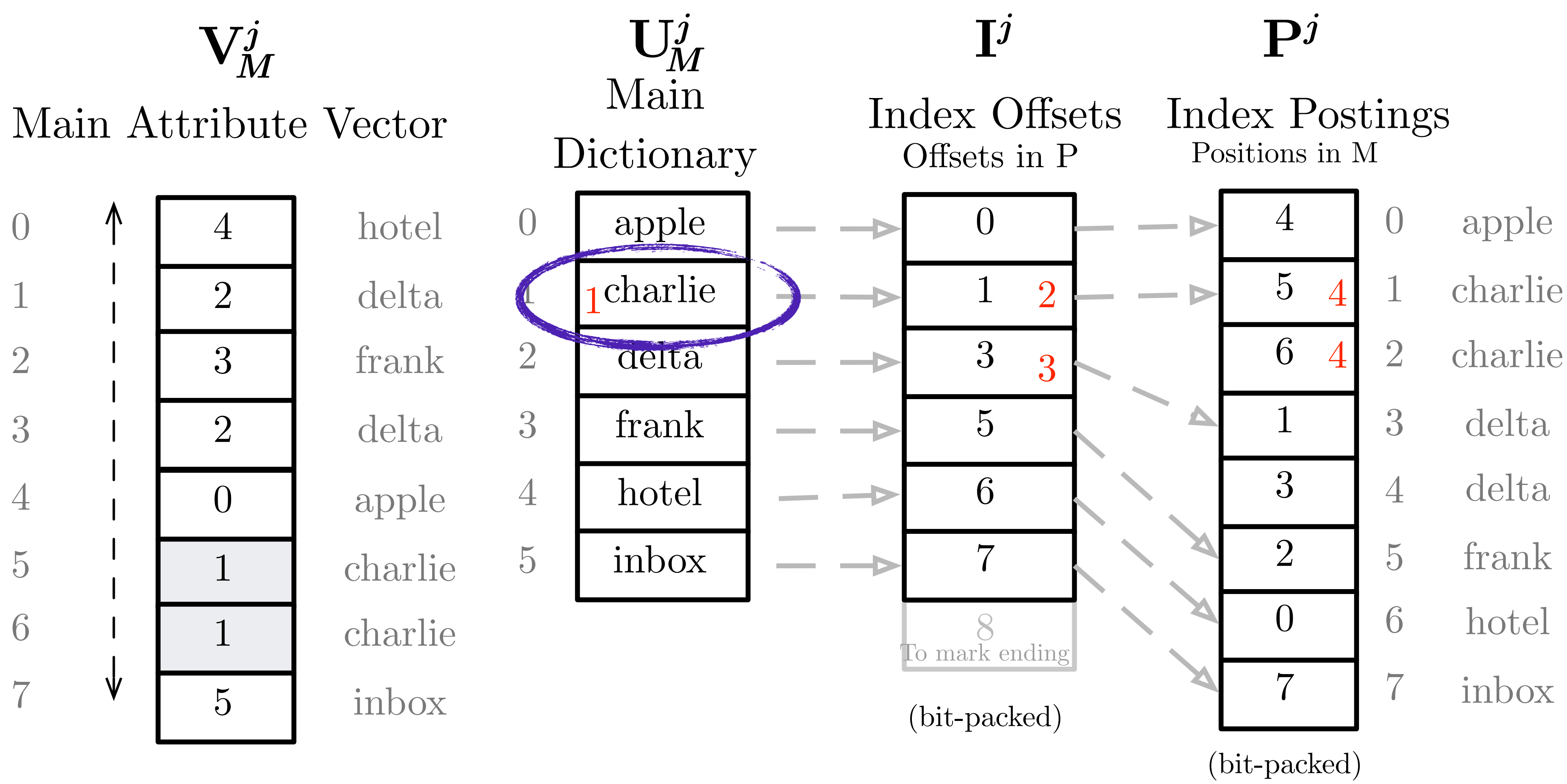
$$\mathbf{E}^j_C = \lceil \log_2 |\mathbf{U}^j_{\mathbf{M}}| \rceil$$

$$\mathbf{A}^j = \lceil \log_2(\mathbf{N}_M) \rceil \text{ bits}$$

$$\textbf{sizeof}(\mathbf{I}^j) = (|\mathbf{U}^j_{\mathbf{M}}| + 1) \cdot \frac{\mathbf{A}^j}{8} \text{ bytes}$$

$$\textbf{sizeof}(\mathbf{P}^j) = \mathbf{N}_M \cdot \frac{\mathbf{A}^j}{8} \text{ bytes}$$

Group-Key Index Structure II



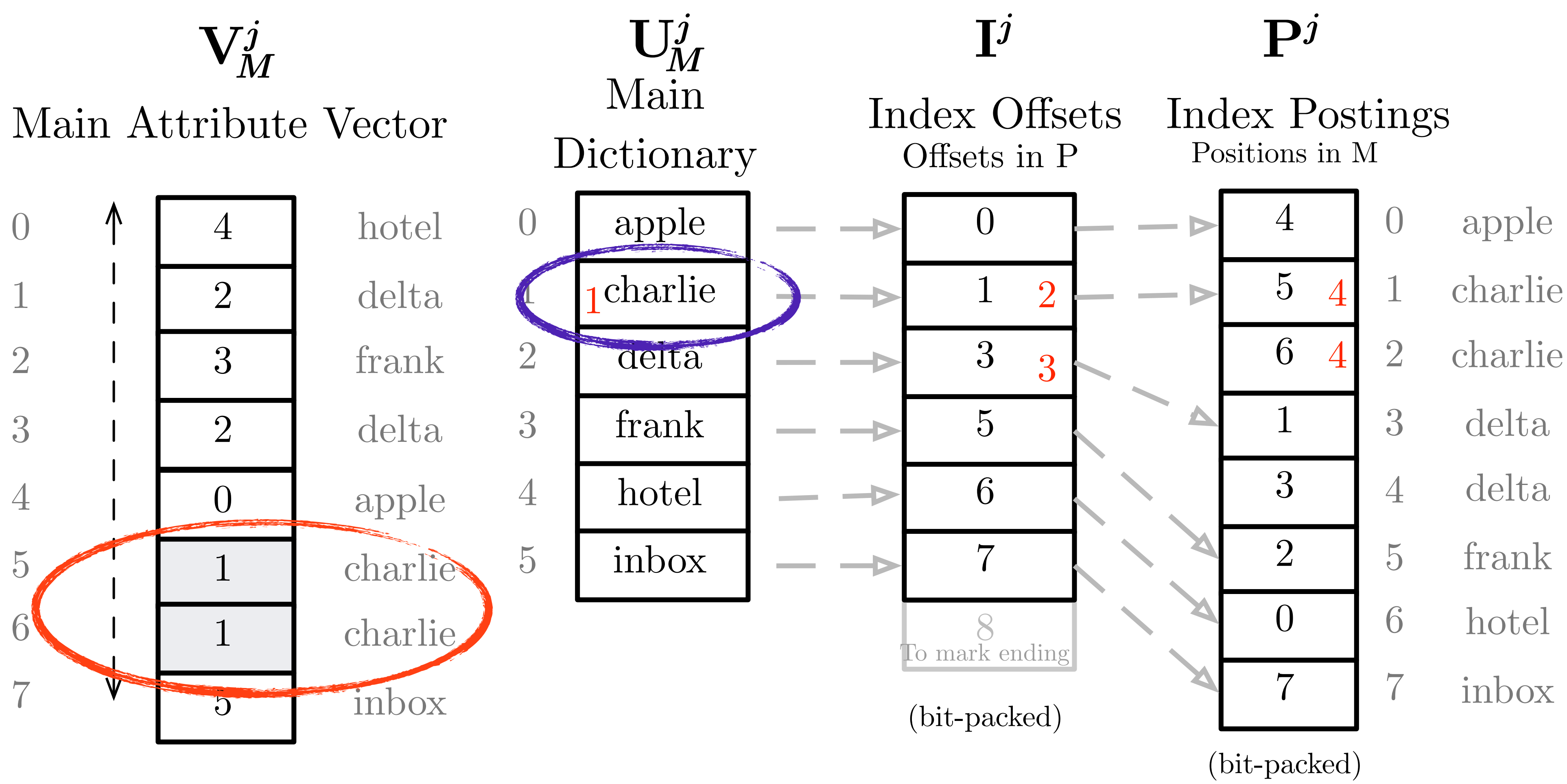
$$\mathbf{E}^j_C = \lceil \log_2 |\mathbf{U}^j_{\mathbf{M}}| \rceil$$

$$\mathbf{A}^j = \lceil \log_2(\mathbf{N}_M) \rceil \text{ bits}$$

$$\mathbf{sizeof(I}^j) = (|\mathbf{U}^j_{\mathbf{M}}| + 1) \cdot \frac{\mathbf{A}^j}{8} \text{ bytes}$$

$$\mathbf{sizeof(P}^j) = \mathbf{N}_M \cdot \frac{\mathbf{A}^j}{8} \text{ bytes}$$

Group-Key Index Structure II



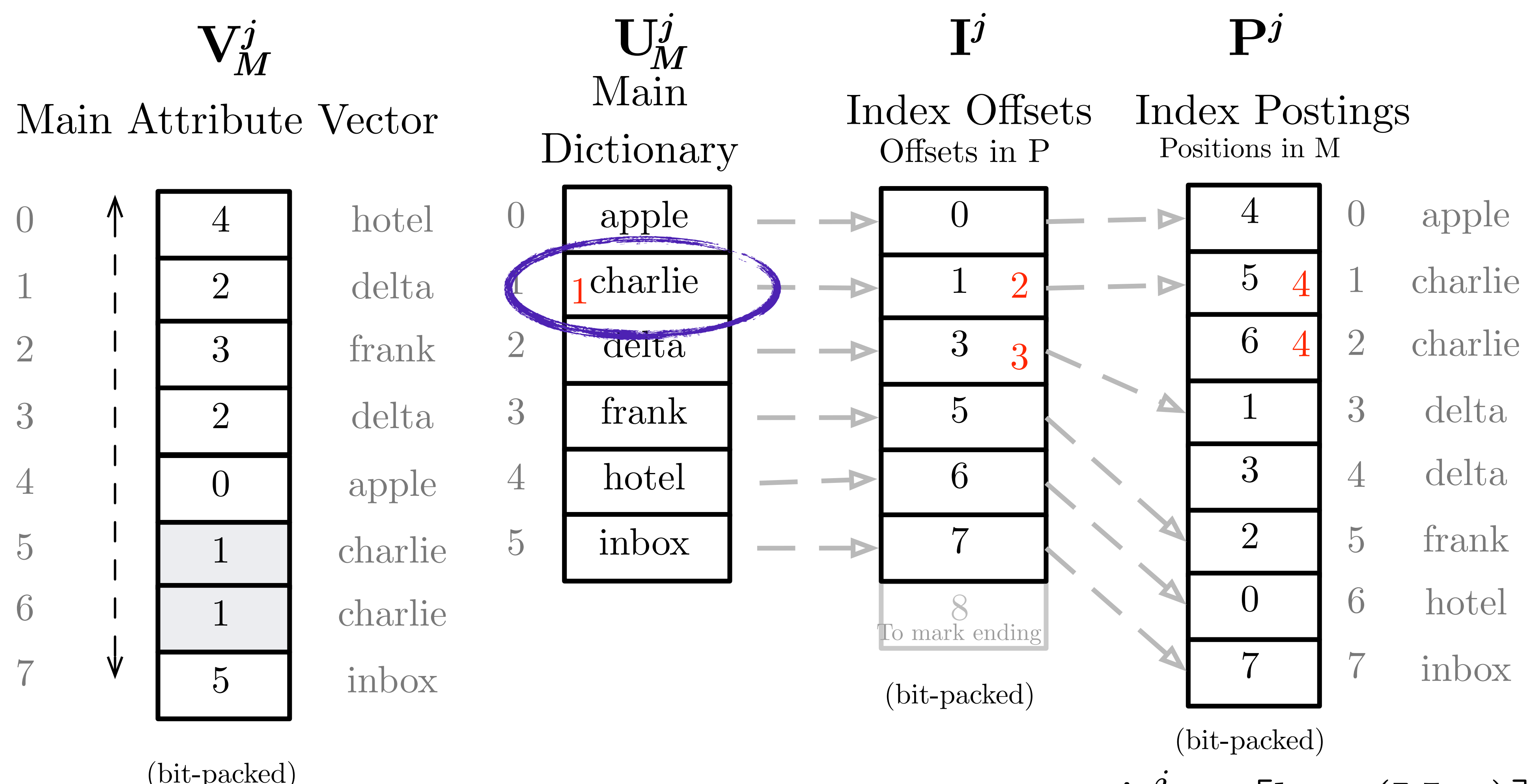
$$\mathbf{E}_C^j = \lceil \log_2 |\mathbf{U}_\mathbf{M}^j| \rceil$$

$$\mathbf{A}^j = \lceil \log_2(\mathbf{N}_M) \rceil \text{ bits}$$

$$\textbf{sizeof}(\mathbf{I}^j) = (|\mathbf{U}_\mathbf{M}^j| + 1) \cdot \frac{\mathbf{A}^j}{8} \text{ bytes}$$

$$\textbf{sizeof}(\mathbf{P}^j) = \mathbf{N}_M \cdot \frac{\mathbf{A}^j}{8} \text{ bytes}$$

Group-Key Index Structure II



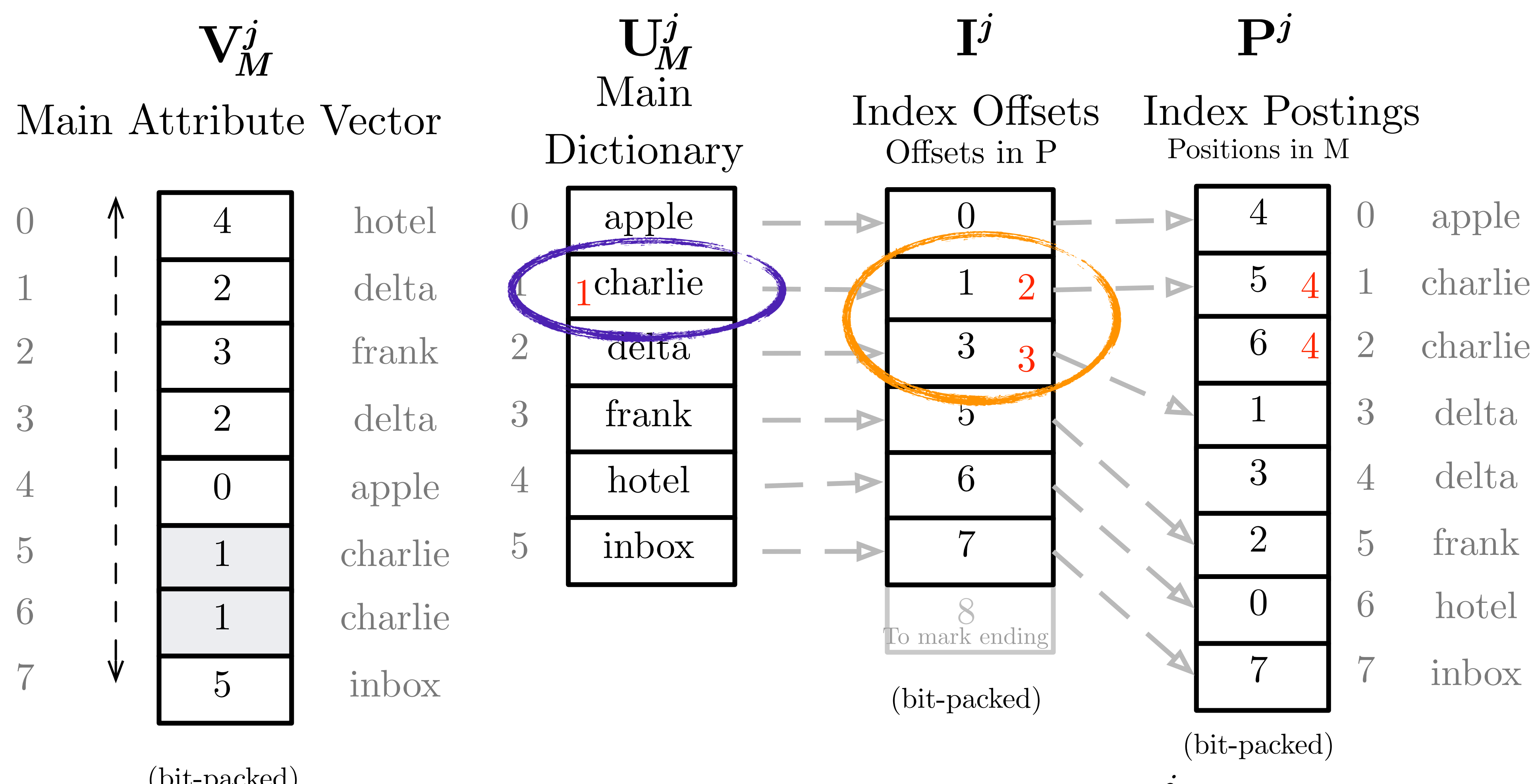
$$\mathbf{E}_C^j = \lceil \log_2 |\mathbf{U}_\mathbf{M}^j| \rceil$$

$$\mathbf{A}^j = \lceil \log_2(\mathbf{N}_M) \rceil \text{ bits}$$

$$\textbf{sizeof}(\mathbf{I}^j) = (|\mathbf{U}_\mathbf{M}^j| + 1) \cdot \frac{\mathbf{A}^j}{8} \text{ bytes}$$

$$\textbf{sizeof}(\mathbf{P}^j) = \mathbf{N}_M \cdot \frac{\mathbf{A}^j}{8} \text{ bytes}$$

Group-Key Index Structure II



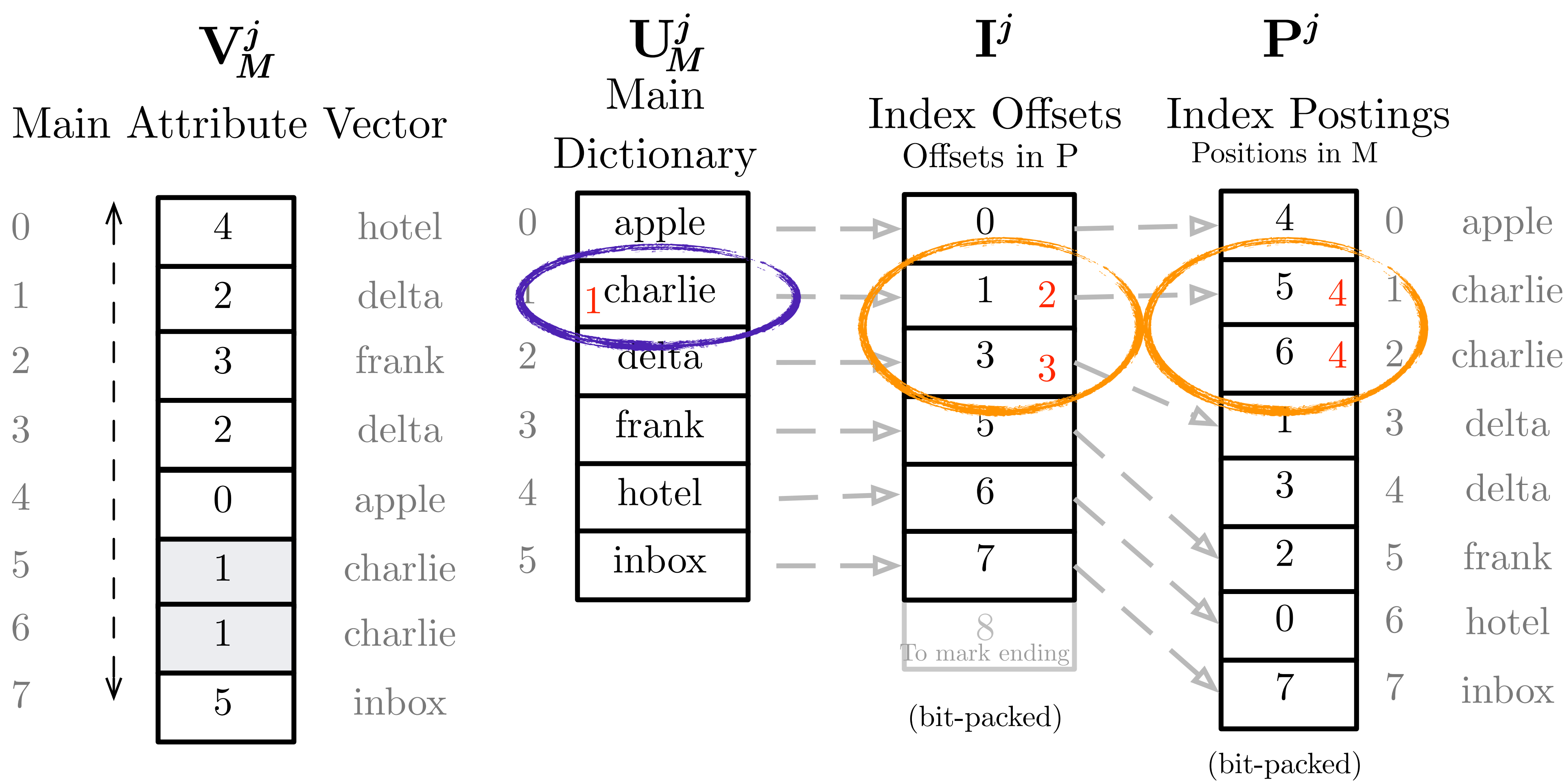
$$\mathbf{E}^j_C = \lceil \log_2 |\mathbf{U}^j_{\mathbf{M}}| \rceil$$

$$\mathbf{A}^j = \lceil \log_2(\mathbf{N}_M) \rceil \text{ bits}$$

$$\textbf{sizeof}(\mathbf{I}^j) = (|\mathbf{U}^j_{\mathbf{M}}| + 1) \cdot \frac{\mathbf{A}^j}{8} \text{ bytes}$$

$$\textbf{sizeof}(\mathbf{P}^j) = \mathbf{N}_M \cdot \frac{\mathbf{A}^j}{8} \text{ bytes}$$

Group-Key Index Structure II



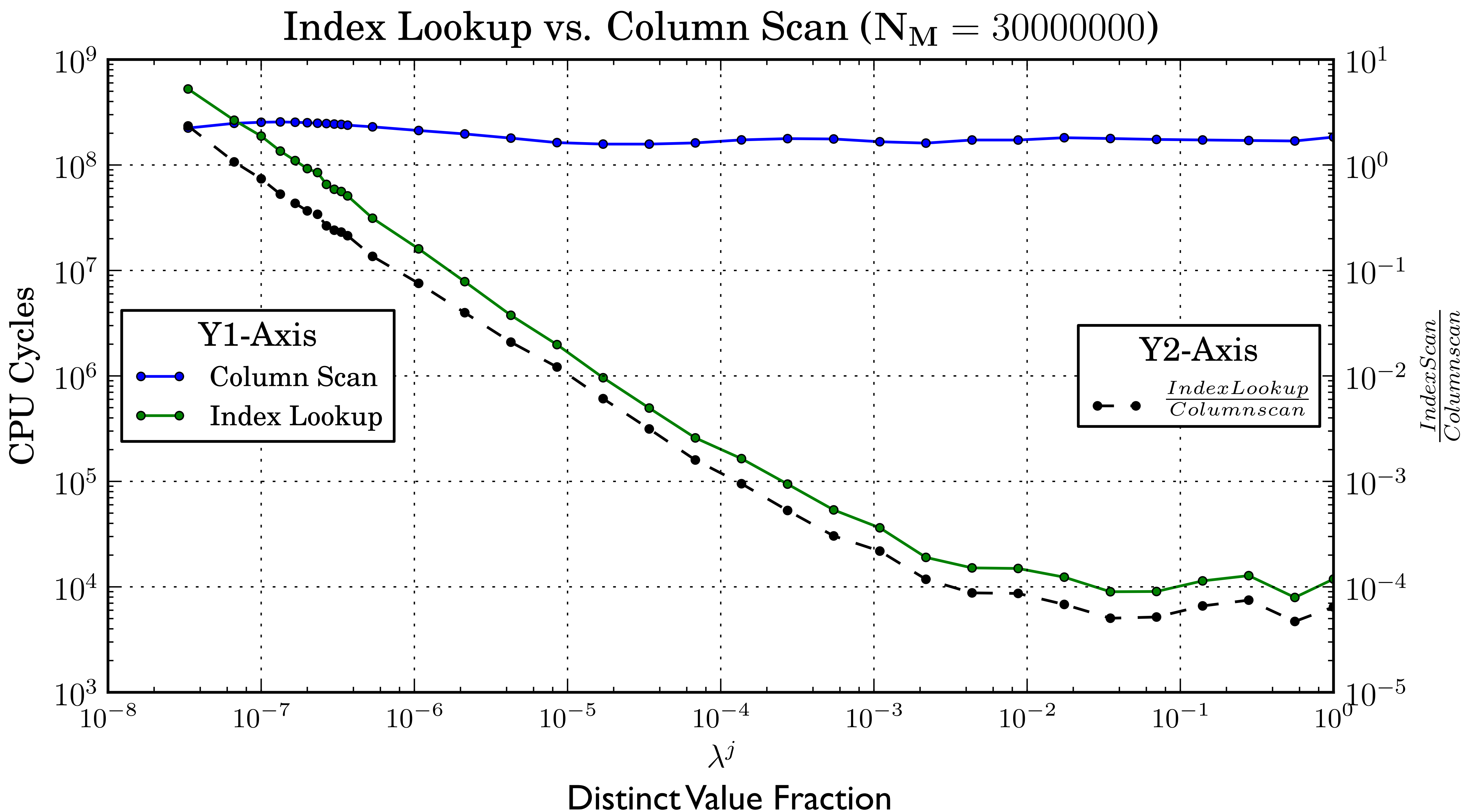
$$\mathbf{E}_C^j = \lceil \log_2 |\mathbf{U}_\mathbf{M}^j| \rceil$$

$$\mathbf{A}^j = \lceil \log_2(\mathbf{N}_M) \rceil \text{ bits}$$

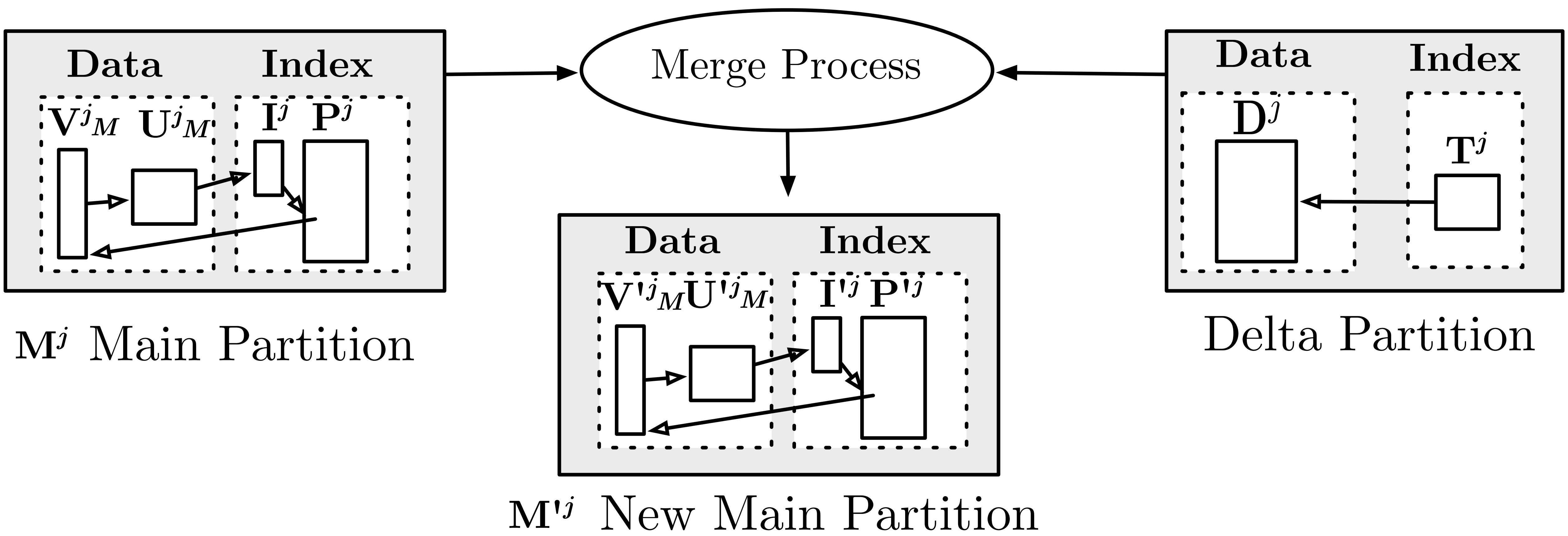
$$\textbf{sizeof}(\mathbf{I}^j) = (|\mathbf{U}_\mathbf{M}^j| + 1) \cdot \frac{\mathbf{A}^j}{8} \text{ bytes}$$

$$\textbf{sizeof}(\mathbf{P}^j) = \mathbf{N}_M \cdot \frac{\mathbf{A}^j}{8} \text{ bytes}$$

Lookup Performance



Merge Process



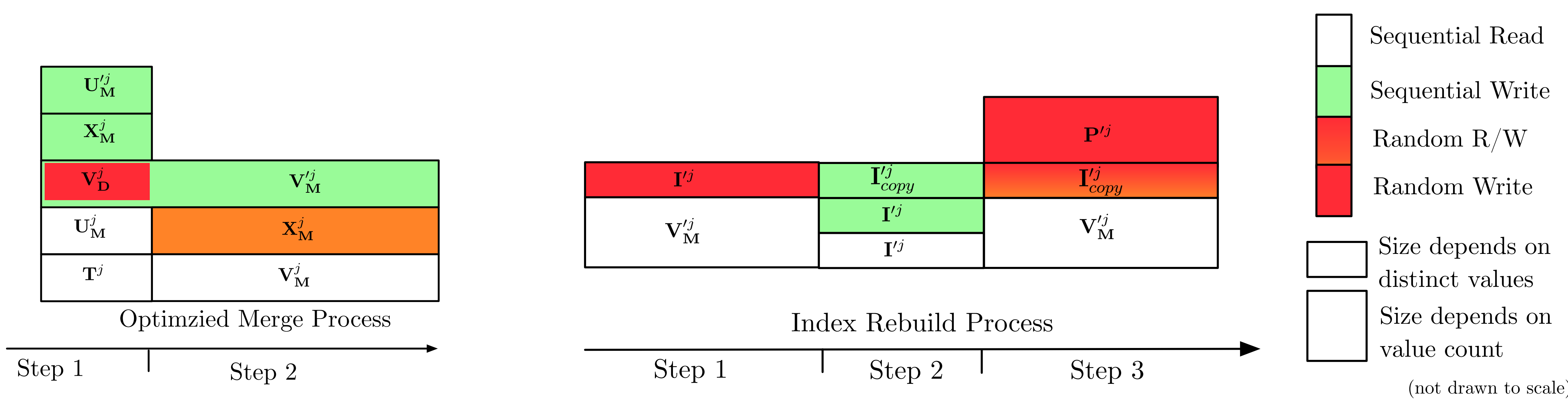
Group-Key Index Creation from Scratch

Column Merge

1. Transform delta partition, merge dictionaries
2. Update main partition's value-ids

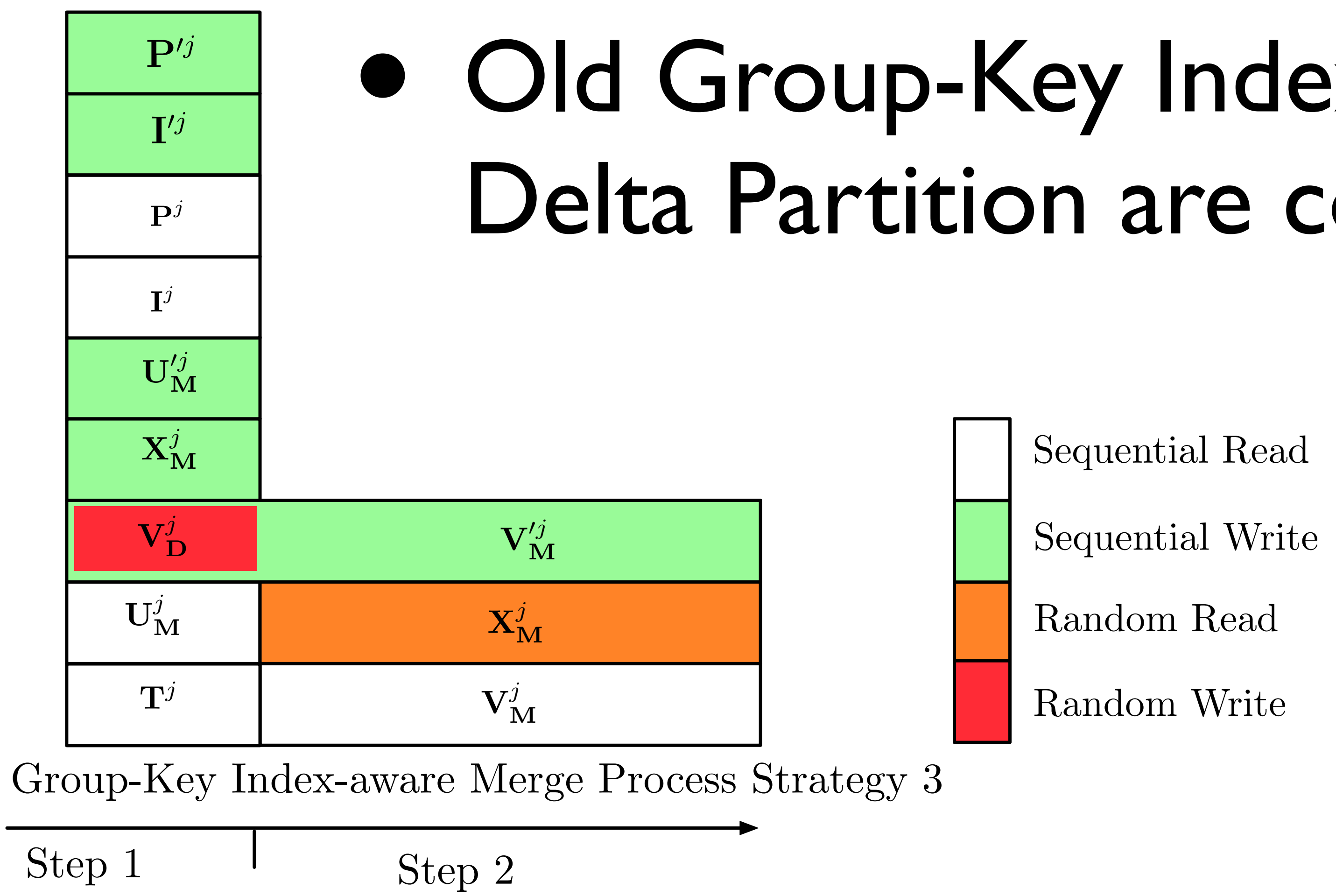
Index Creation

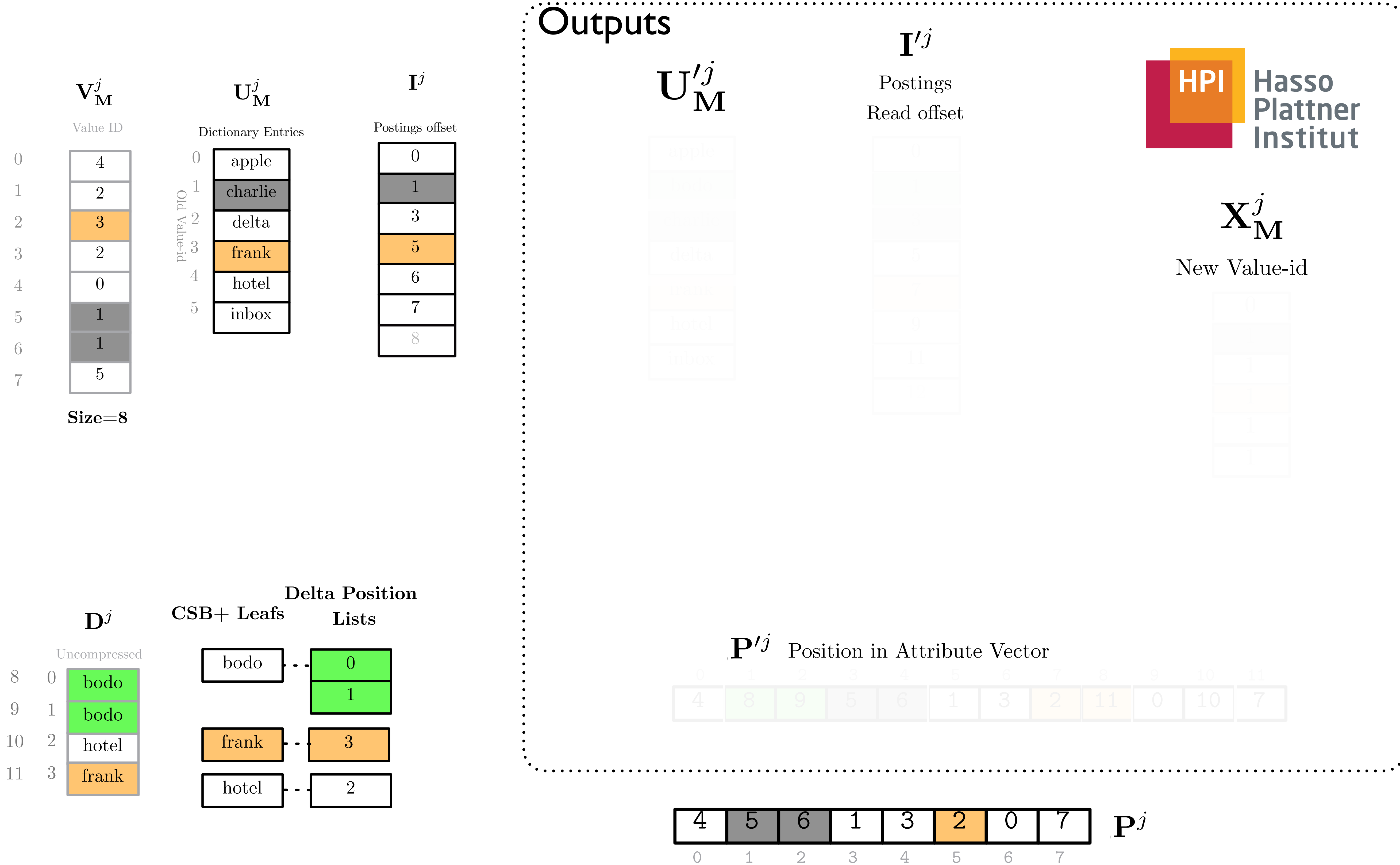
1. Counting occurrences of value-ids
2. Creating offsets for **P** in **I**
3. Creating postings in **P**



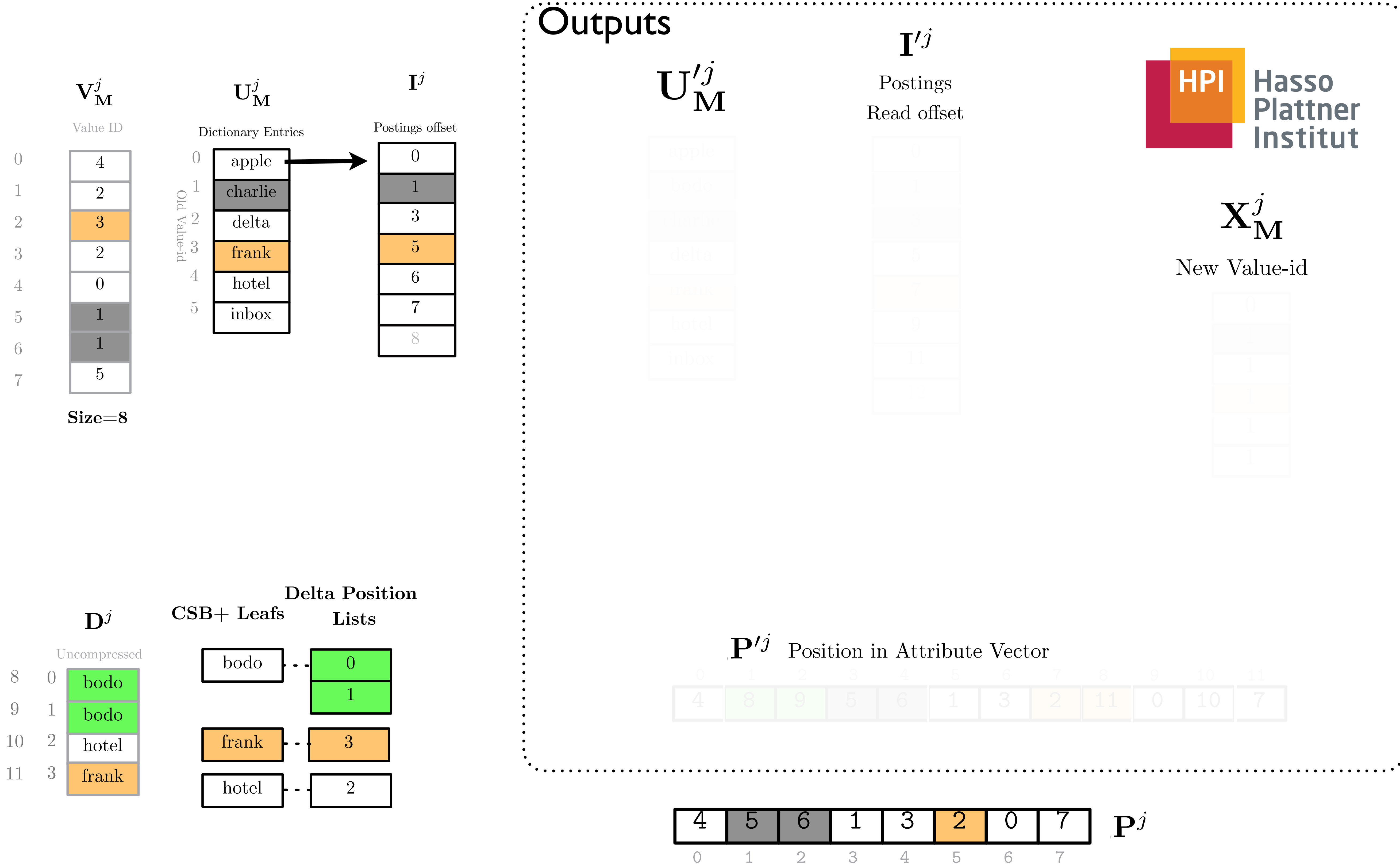
Integrate with Merge Process

- Merge index together with dictionaries
- Old Group-Key Index and CSB+ Index on Delta Partition are combined





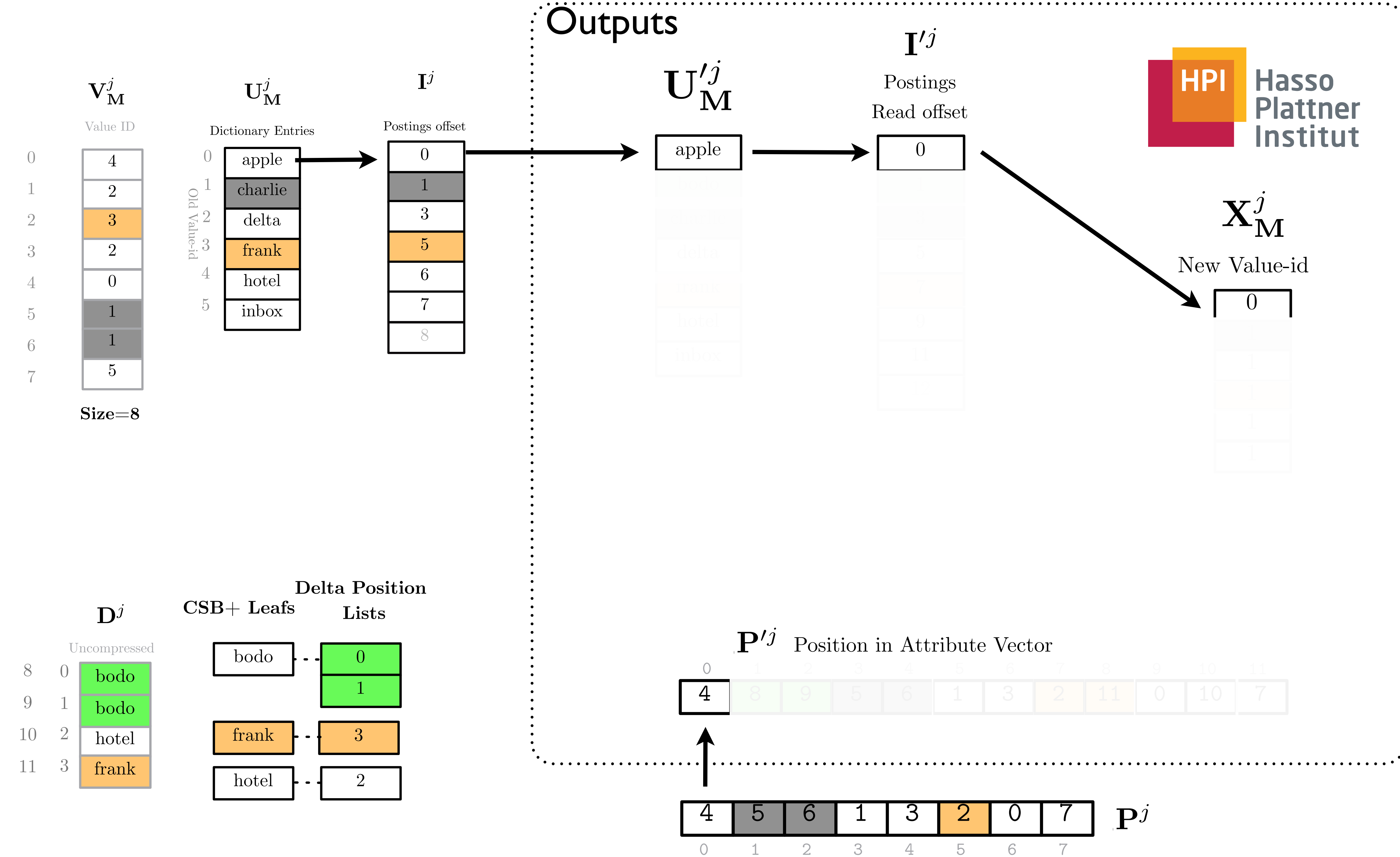
C (corresponding tuples) = **0 1 3 5 7 9 11 12**



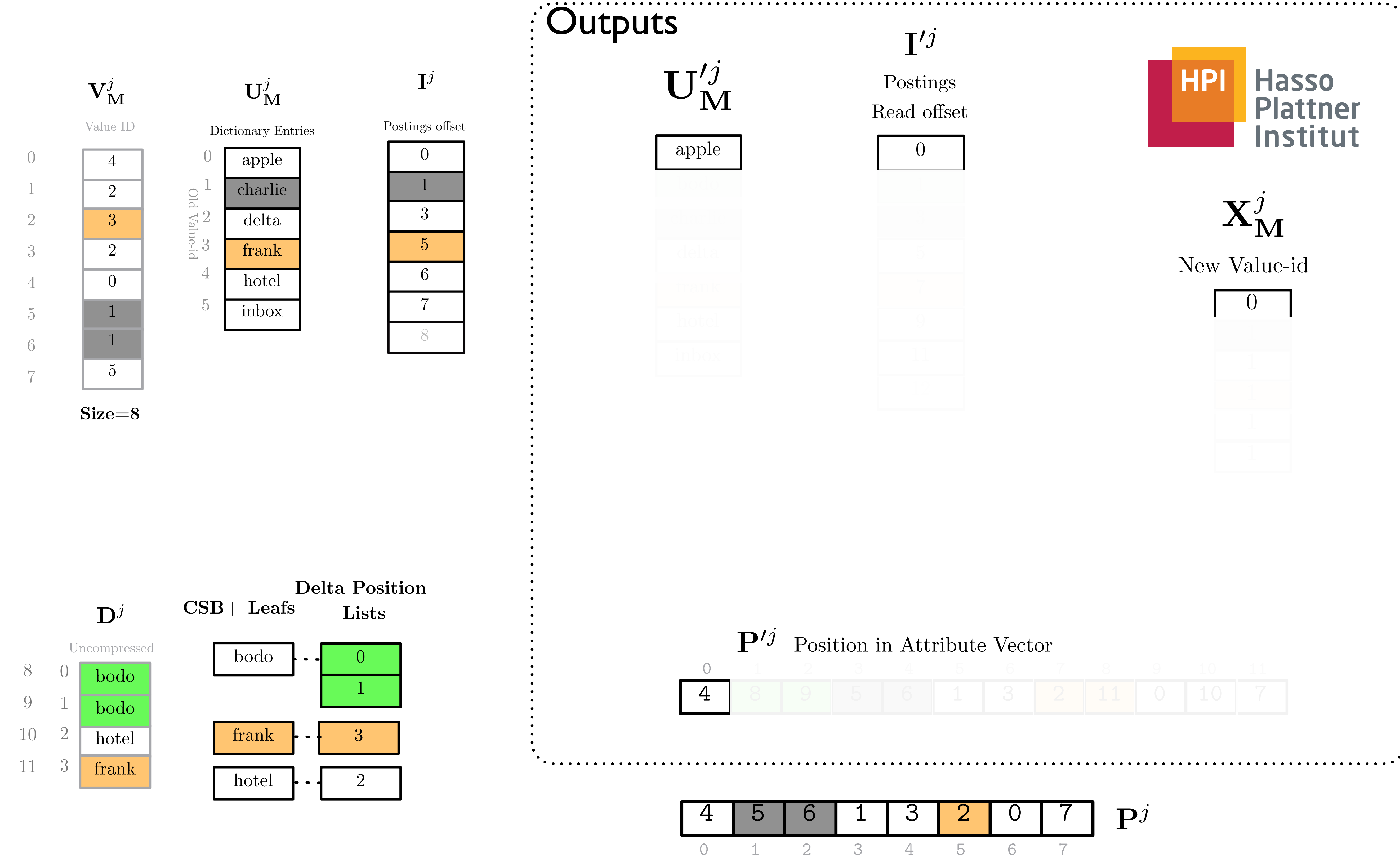
C (corresponding tuples) = **0 1 3 5 7 9 11 12**

Martin Faust | ADMS 2012

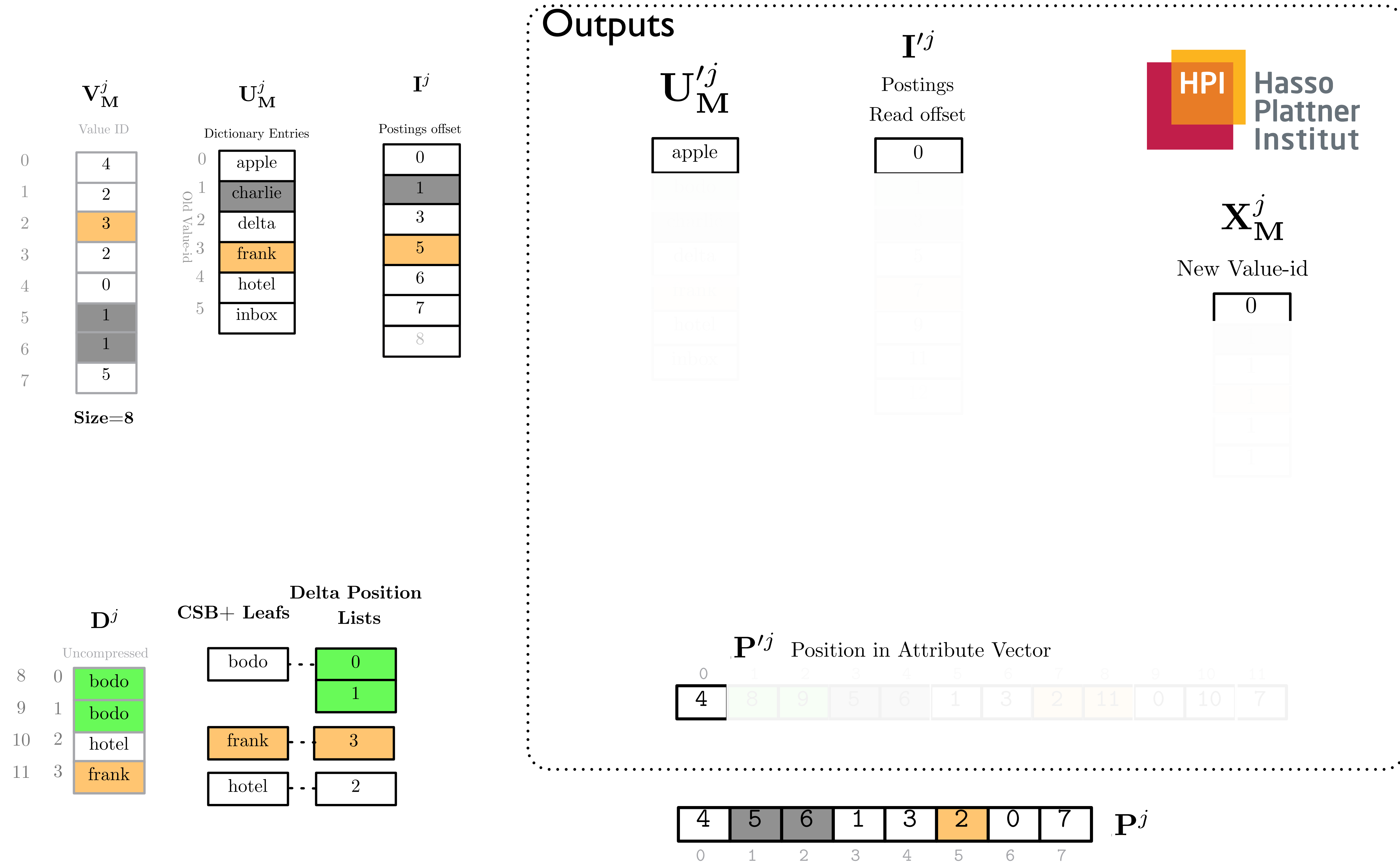
Monday, August 27, 12



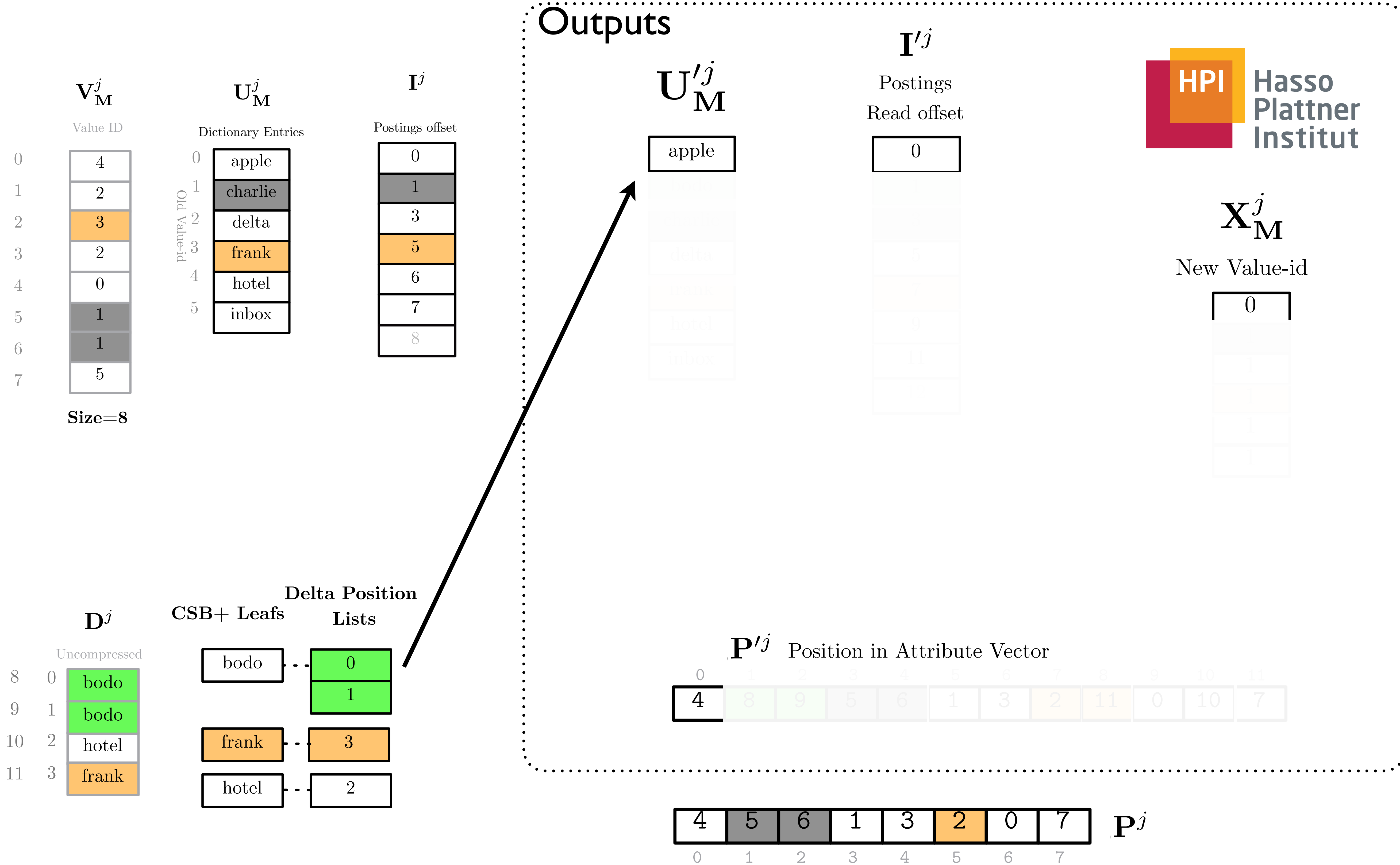
C (corresponding tuples) = **0 1 3 5 7 9 11 12**



C (corresponding tuples) = 0 1 3 5 7 9 11 12

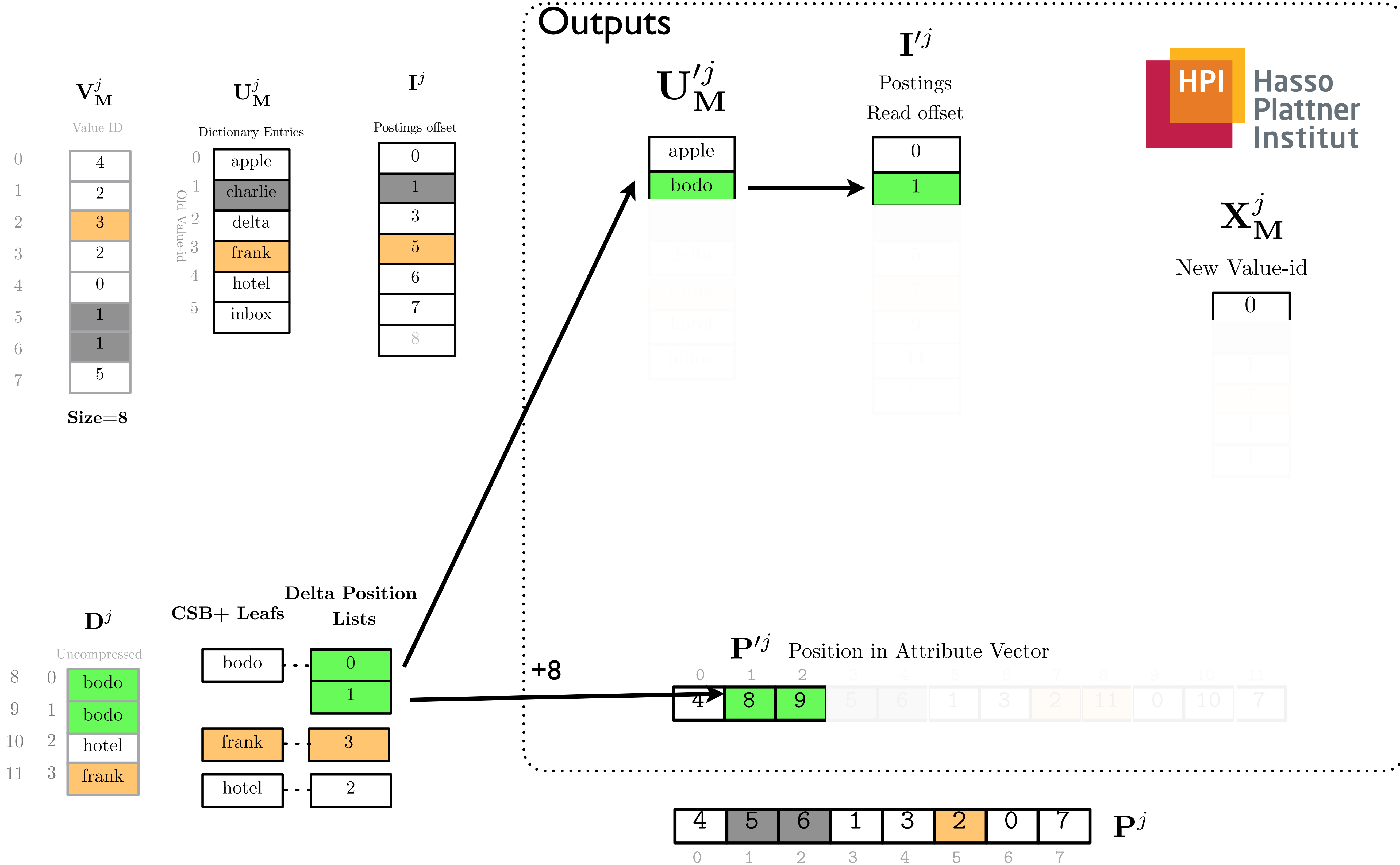


C (corresponding tuples) = **0 1 3 5 7 9 11 12**

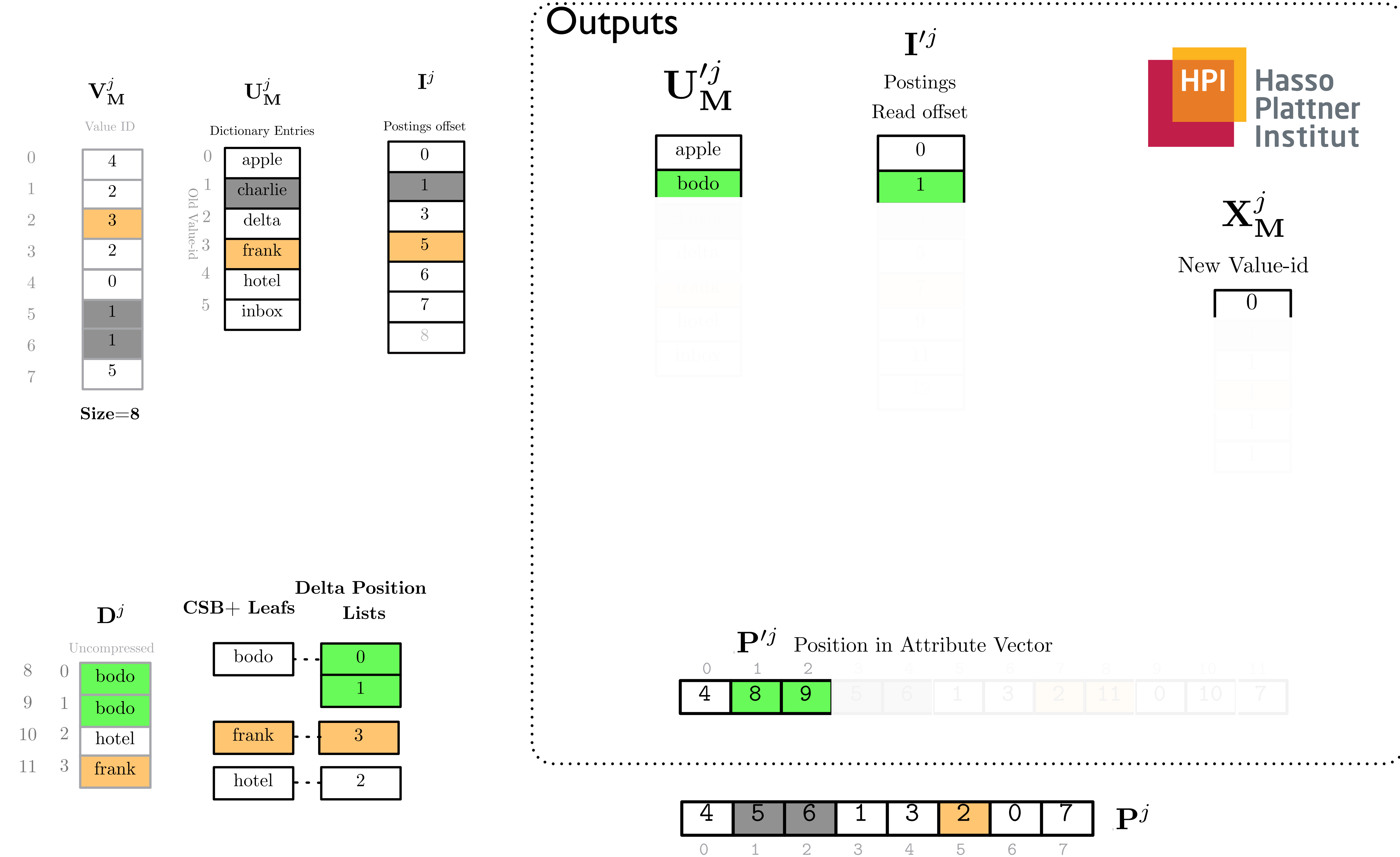


C (corresponding tuples) = **0 1 3 5 7 9 11 12**

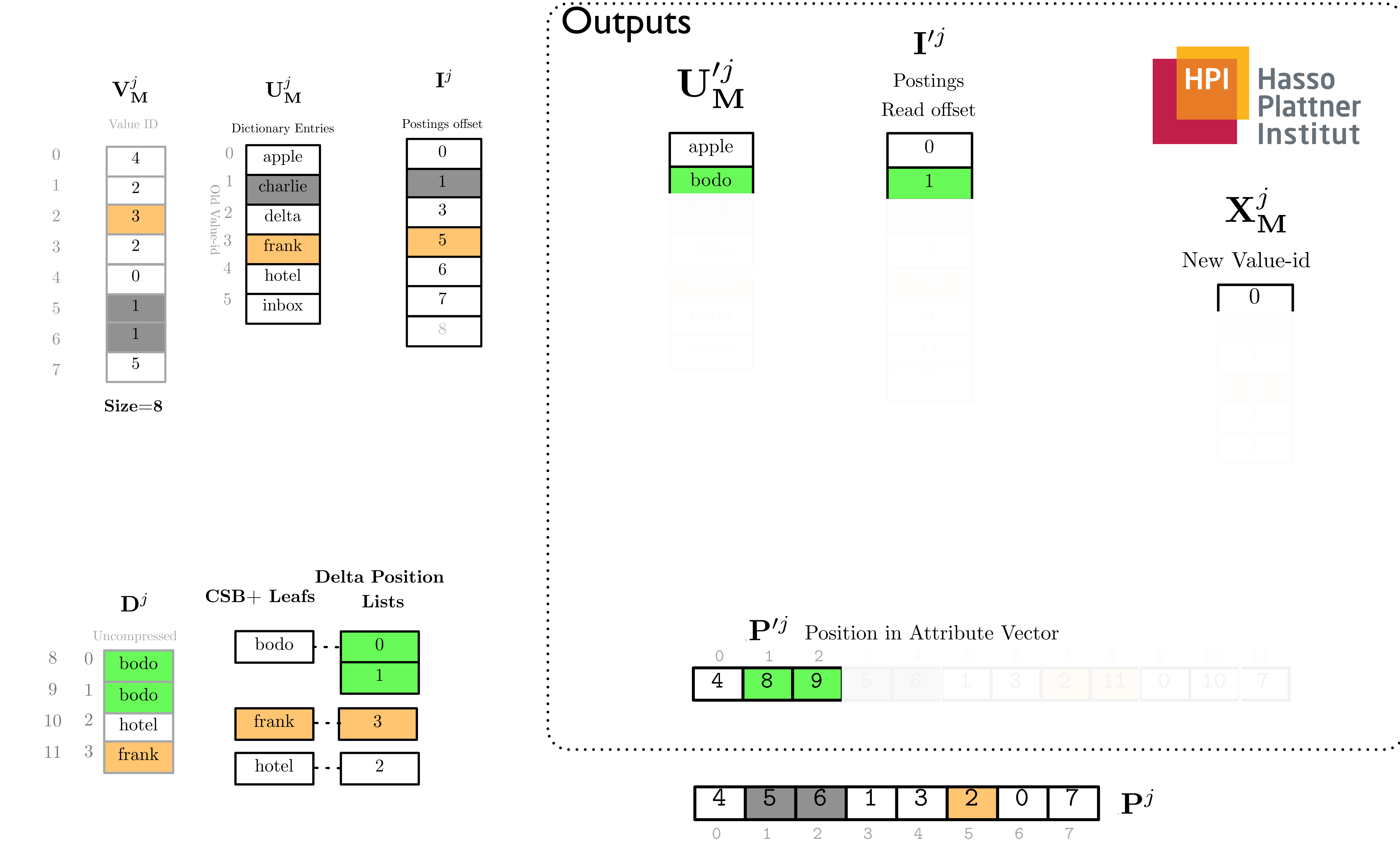
Martin Faust | ADMS 2012



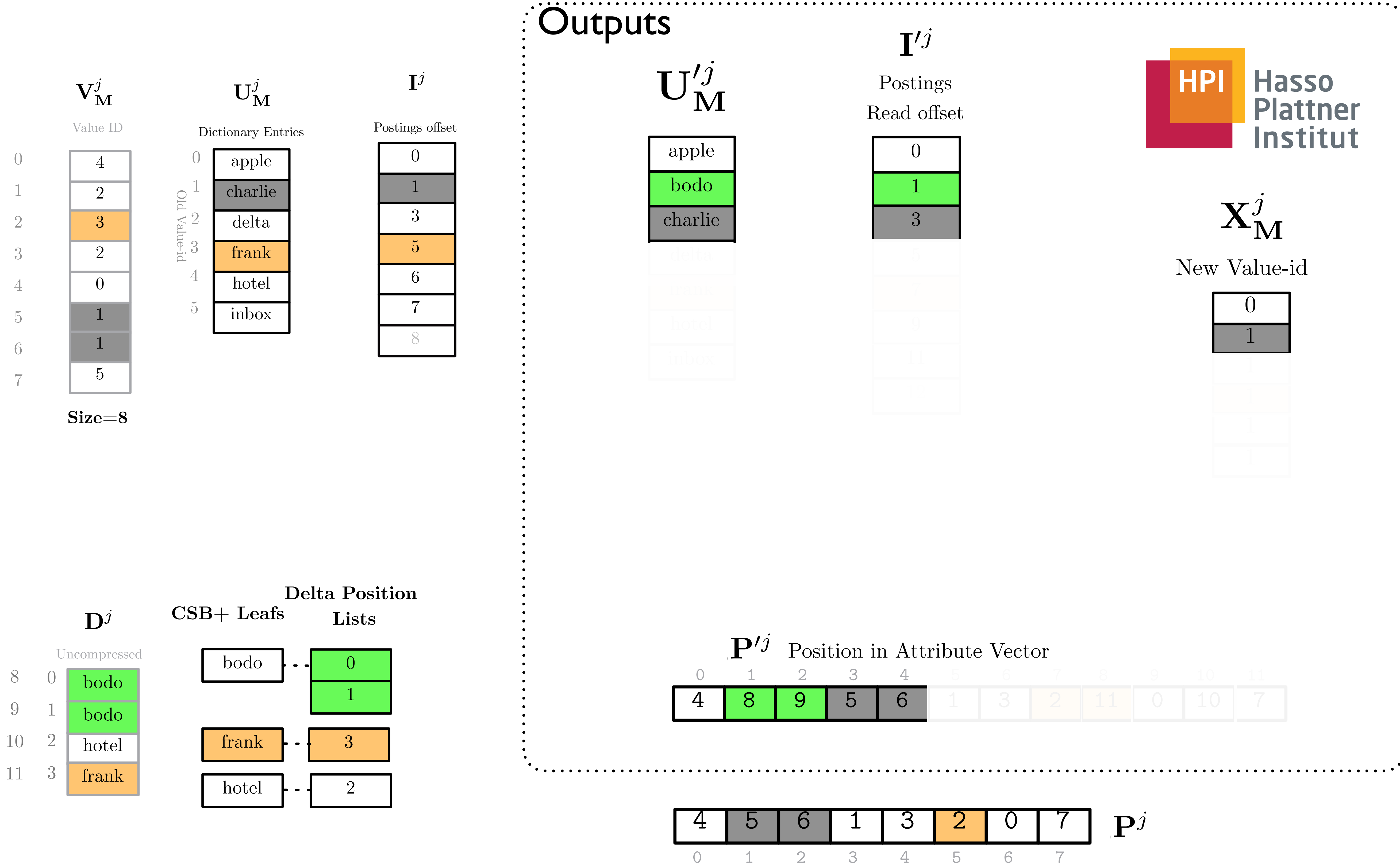
C (corresponding tuples) = **0 1 3 5 7 9 11 12**



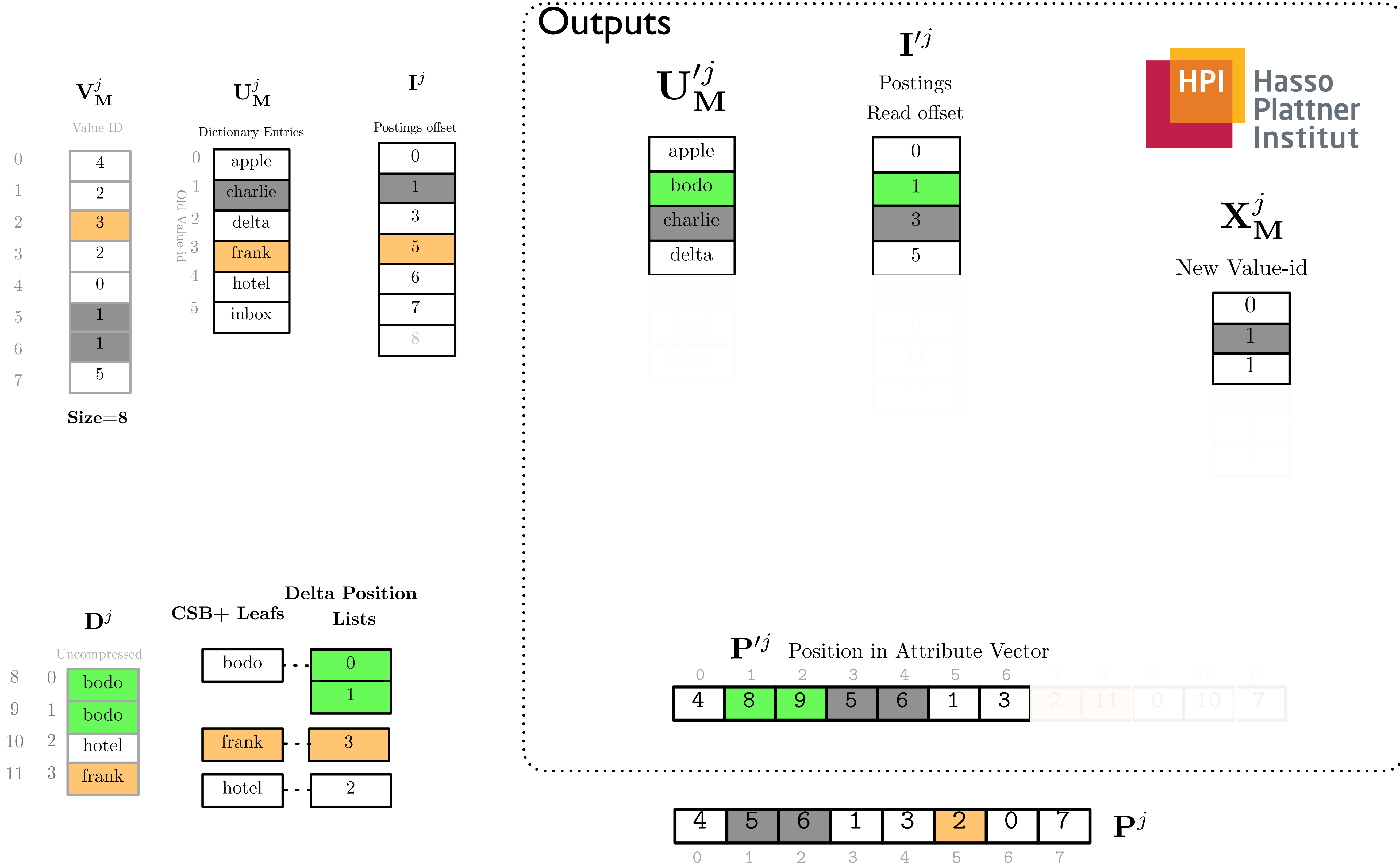
C (corresponding tuples) = **0 1 3 5 7 9 11 12**



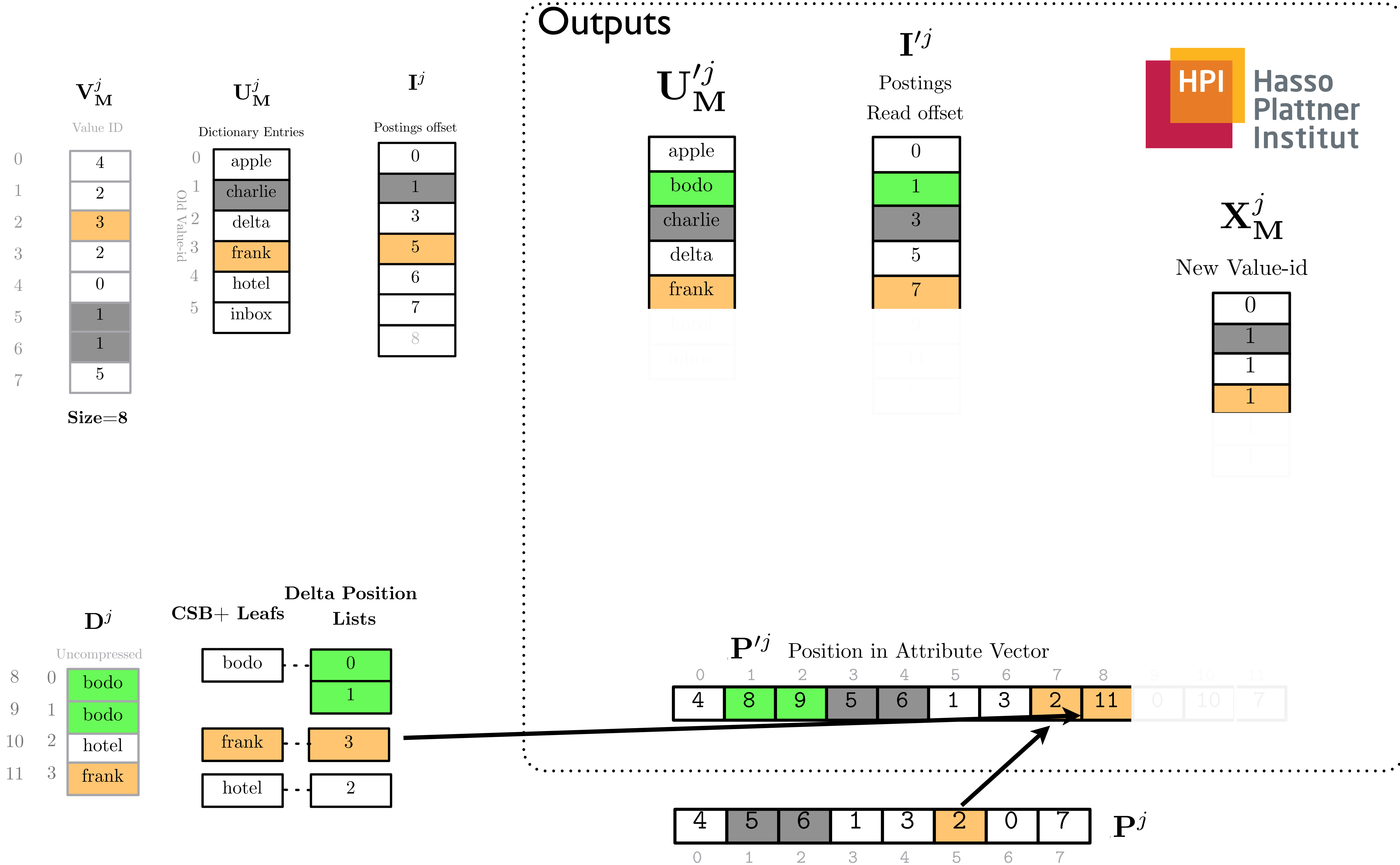
C (corresponding tuples) = **0 1 3 5 7 9 11 12**



C (corresponding tuples) = **0 1 3 5 7 9 11 12**



C (corresponding tuples) = **0 1 3 5 7 9 11 12**



C (corresponding tuples) = **0 1 3 5 7 9 11 12**

	$\mathbf{V}_\mathbf{M}^j$ Value ID
0	4
1	2
2	3
3	2
4	0
5	1
6	1
7	5

Size=8

Old Value-id	$\mathbf{U}_\mathbf{M}^j$ Dictionary Entries	$\mathbf{I}^j$ Postings offset
0	apple	0
1	charlie	1
2	delta	3
3	frank	5
4	hotel	6
5	inbox	7
		8

		$\mathbf{D}^j$ Uncompressed
8	0	bodo
9	1	bodo
10	2	hotel
11	3	frank

CSB+ Leafs	**Delta Position Lists**
bodo	0, 1
frank	3
hotel	2

Outputs

$\mathbf{U}_\mathbf{M}^{\prime j}$	$\mathbf{I}^{\prime j}$ Postings Read offset
apple	0
bodo	1
charlie	3
delta	5
frank	7
hotel	9
inbox	11

$\mathbf{X}_\mathbf{M}^j$ New Value-id

0
1
1
1

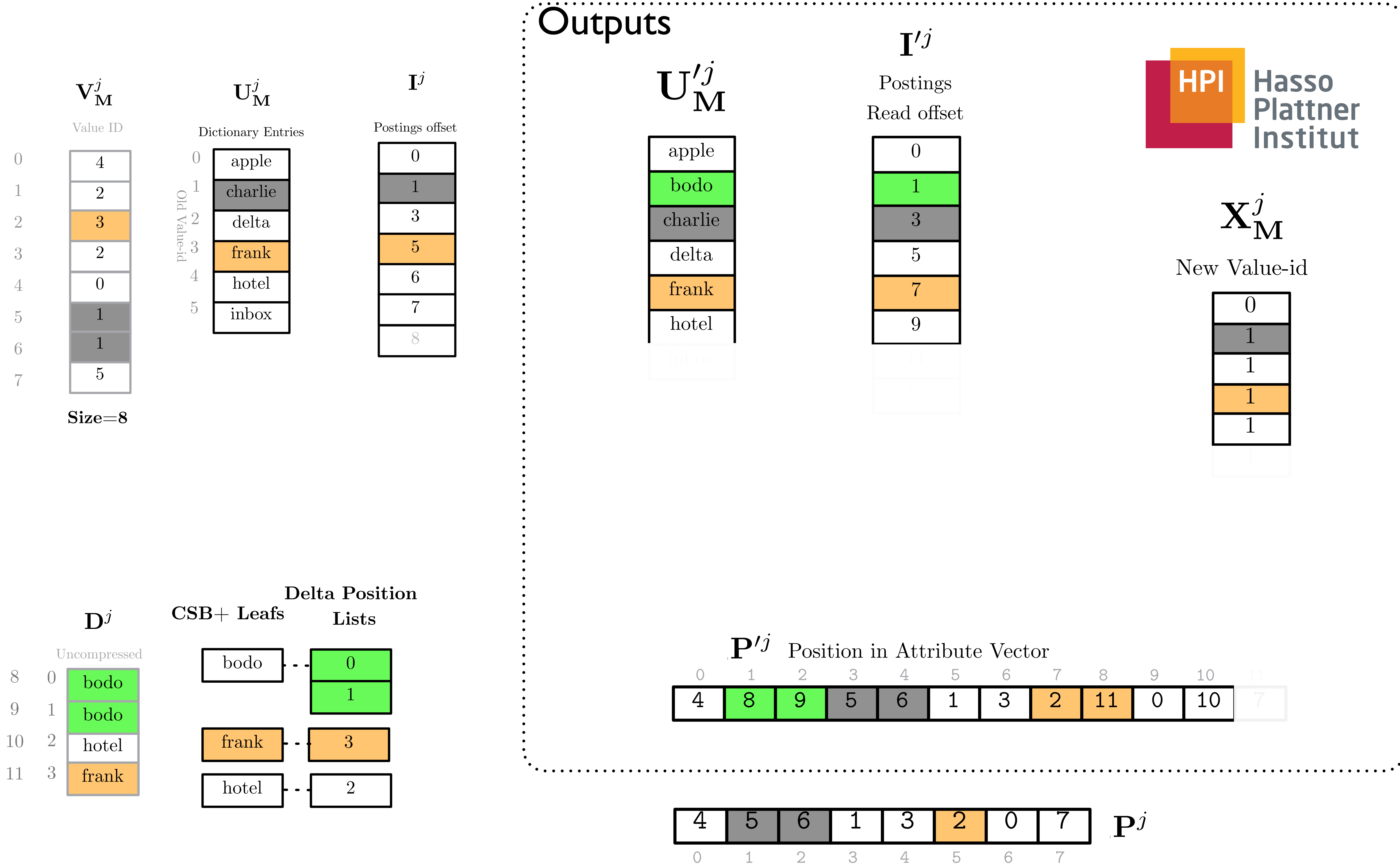
$\mathbf{P}^{\prime j}$ Position in Attribute Vector

0	1	2	3	4	5	6	7	8	9	10	11
4	8	9	5	6	1	3	2	11	0	10	7

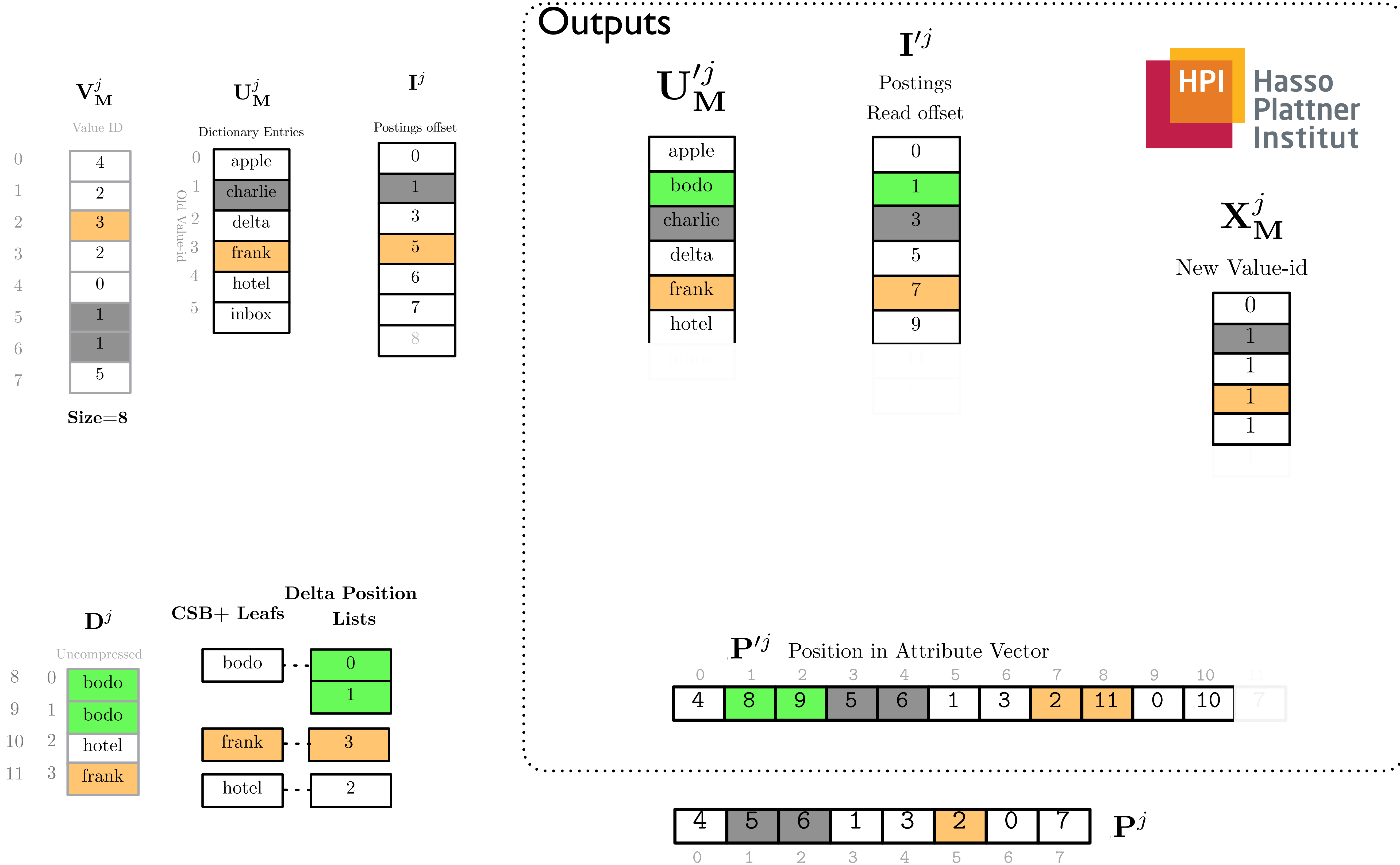
$\mathbf{P}^j$

0	1	2	3	4	5	6	7
4	5	6	1	3	2	0	7

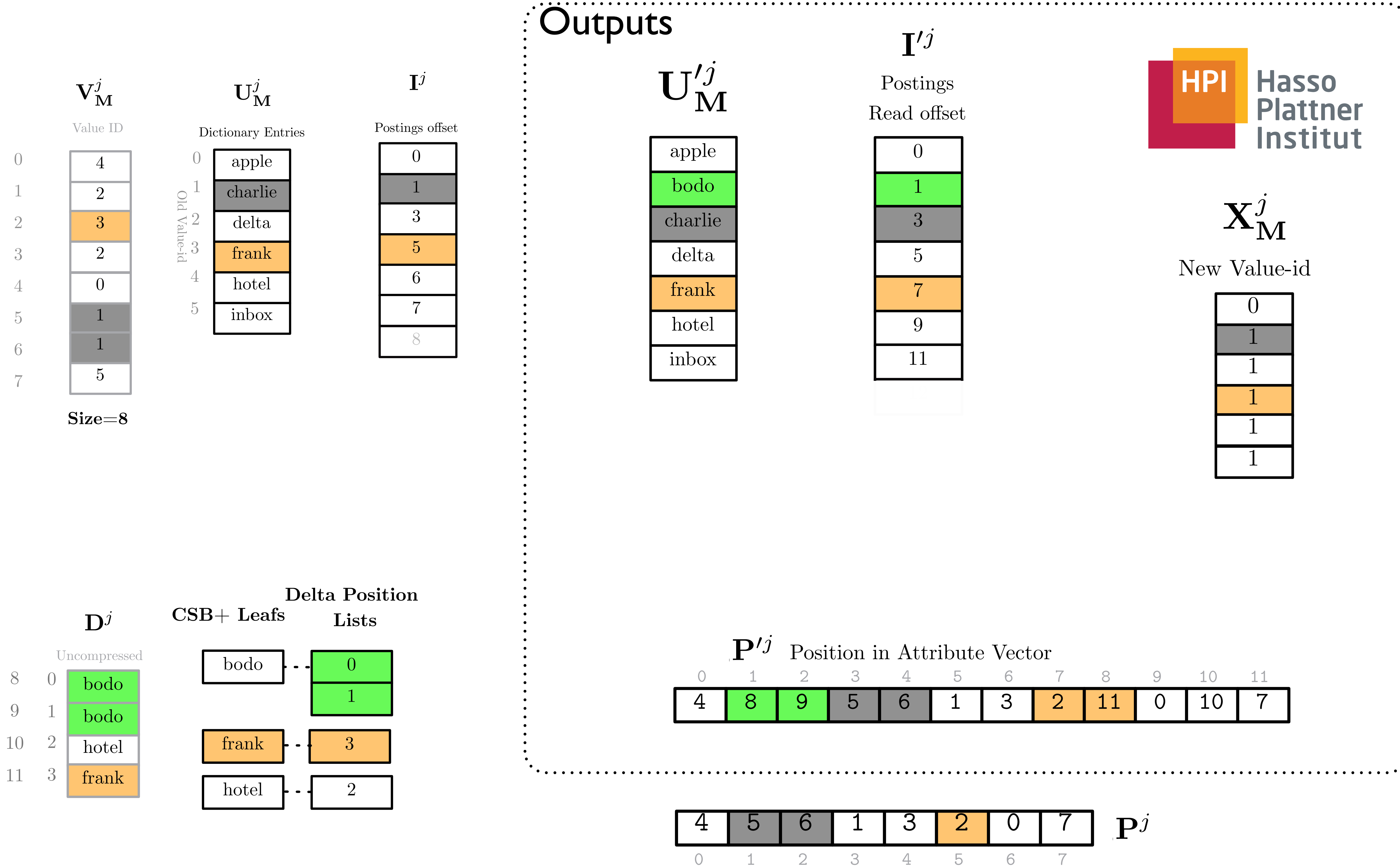
C (corresponding tuples) = **0 1 3 5 7 9 11 12**



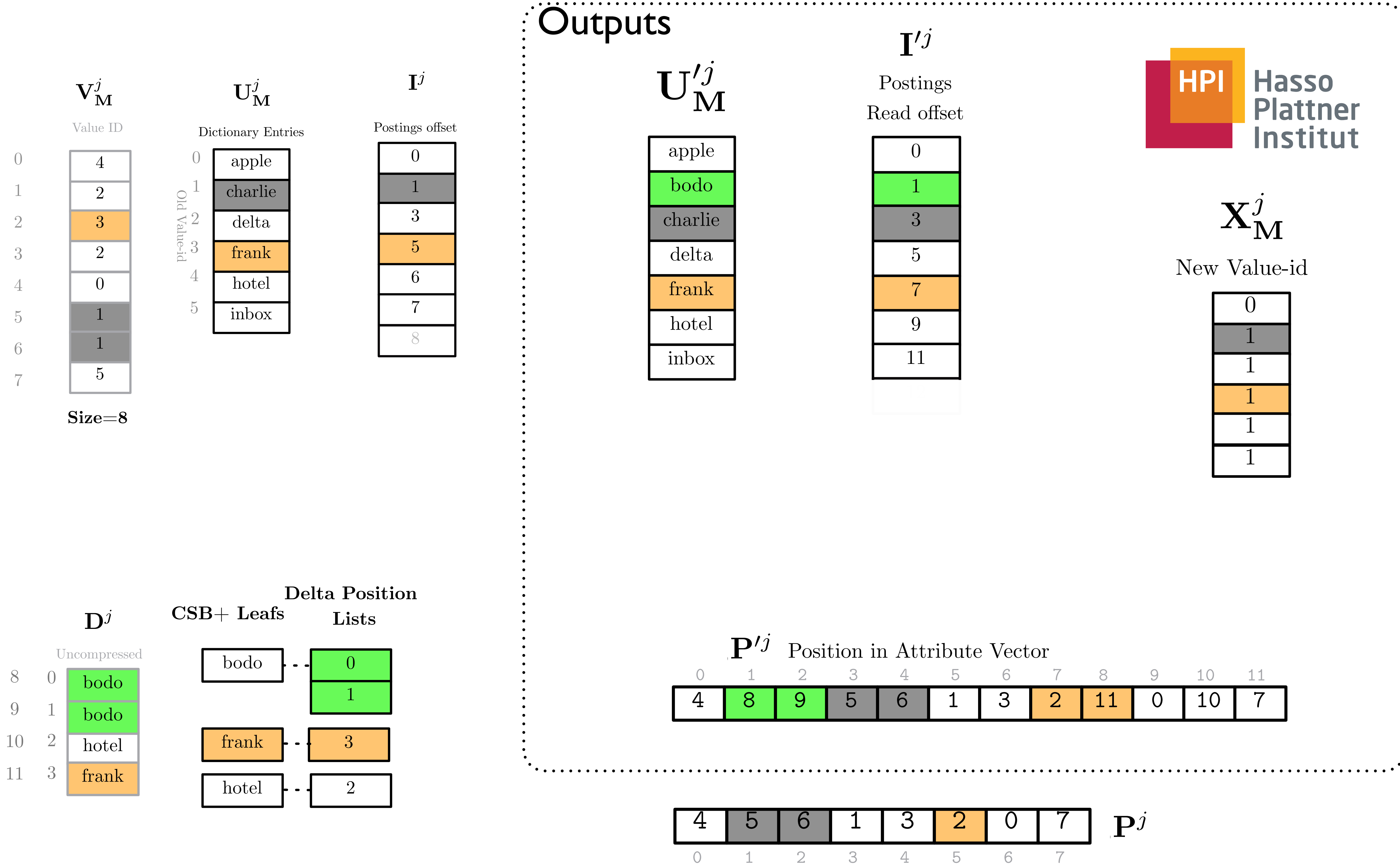
C (corresponding tuples) = **0 1 3 5 7 9 11 12**



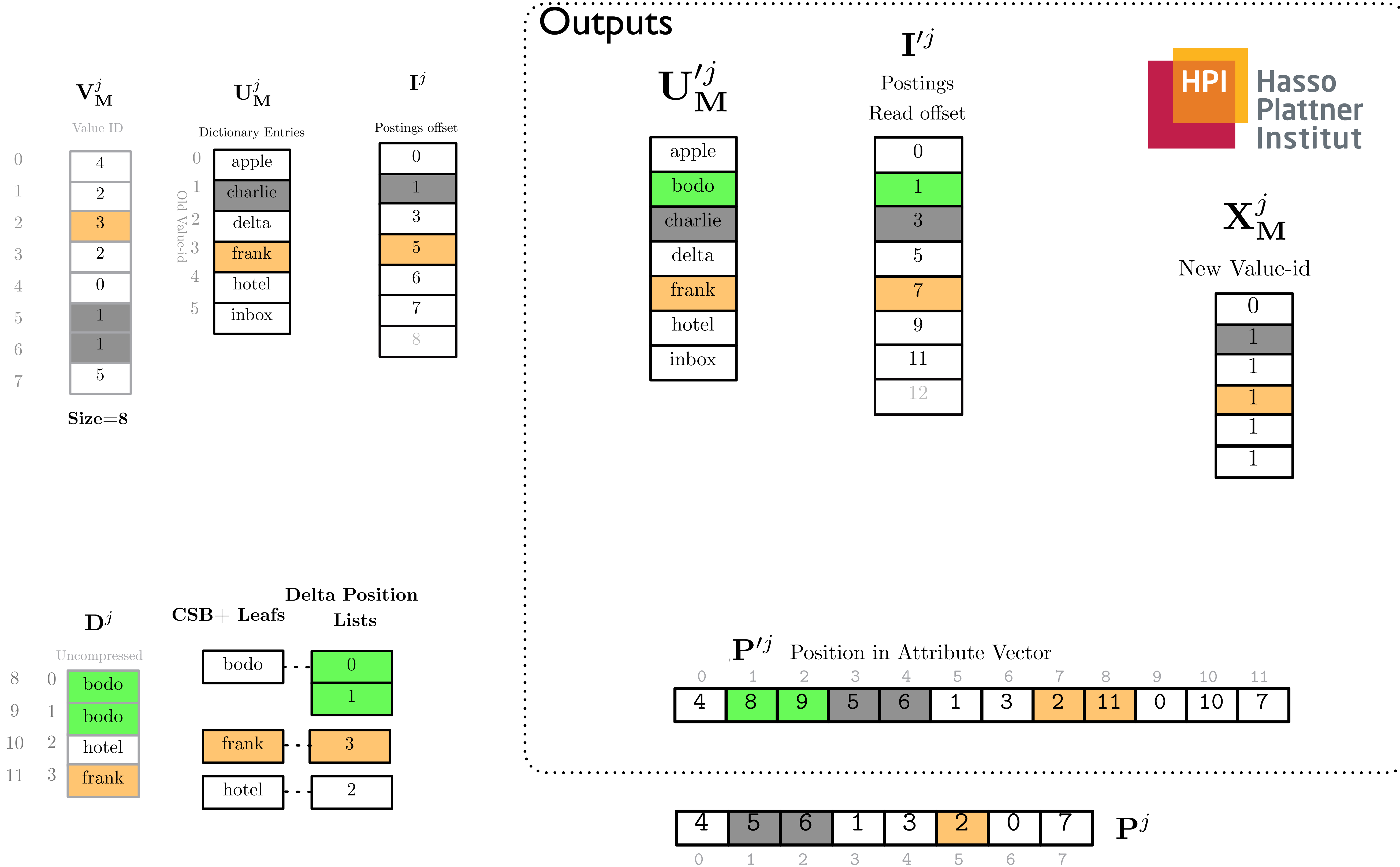
C (corresponding tuples) = 0 1 3 5 7 9 11 12



C (corresponding tuples) = **0 1 3 5 7 9 11 12**



C (corresponding tuples) = 0 1 3 5 7 9 11 12



C (corresponding tuples) = **0 1 3 5 7 9 11 12**

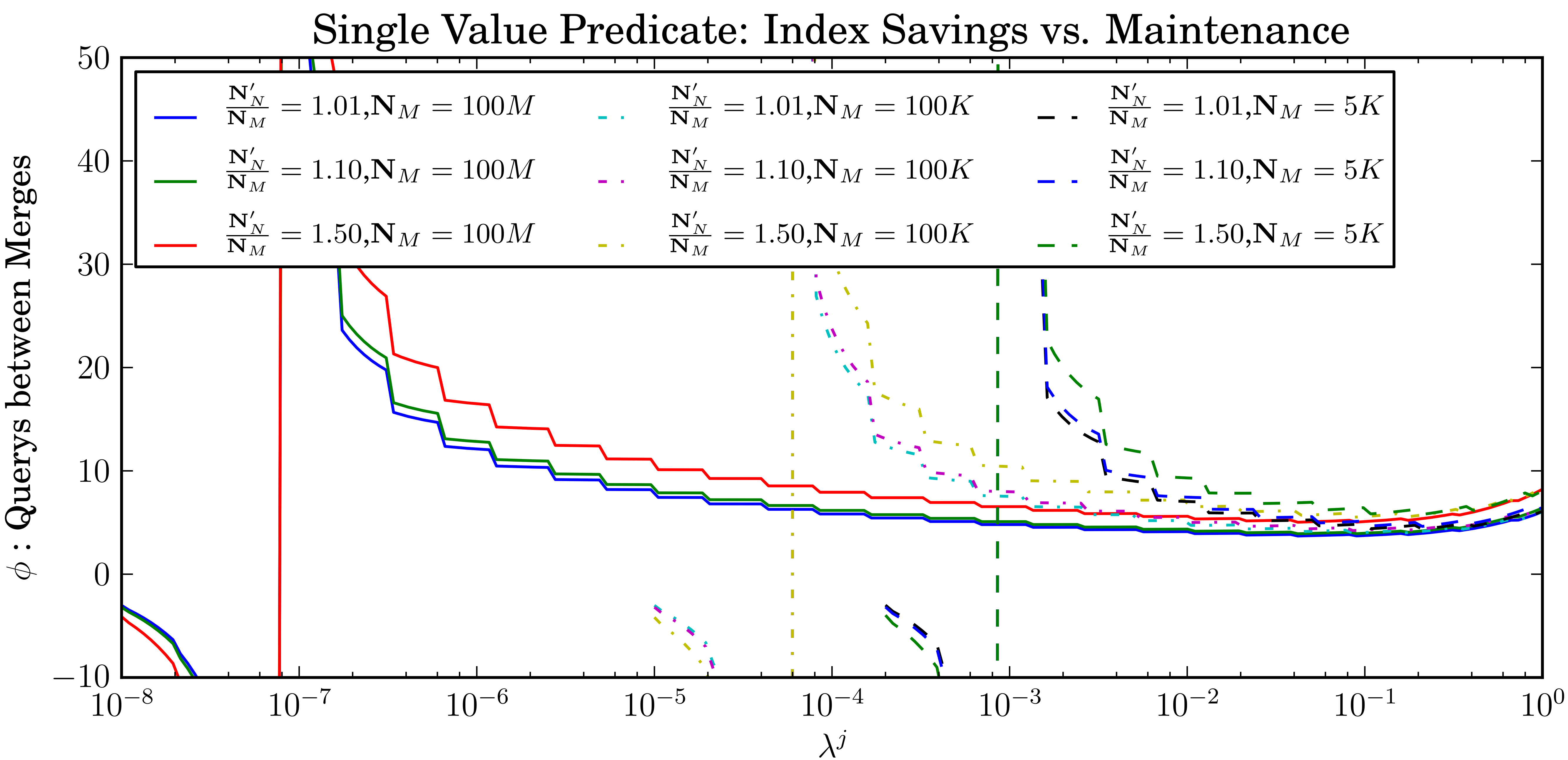
Extra Memory Costs

- Maintaining the index increases costs
- The old index has to be read and the new index has to be written

Offset Increased Memory Costs

- Index decreases memory traffic during query execution by turning column scans into lookups
- Index maintenance increases memory traffic of merge process
- Break-Even point depends on column parameters
- **Savings of how many queries offset the increased merge costs?**

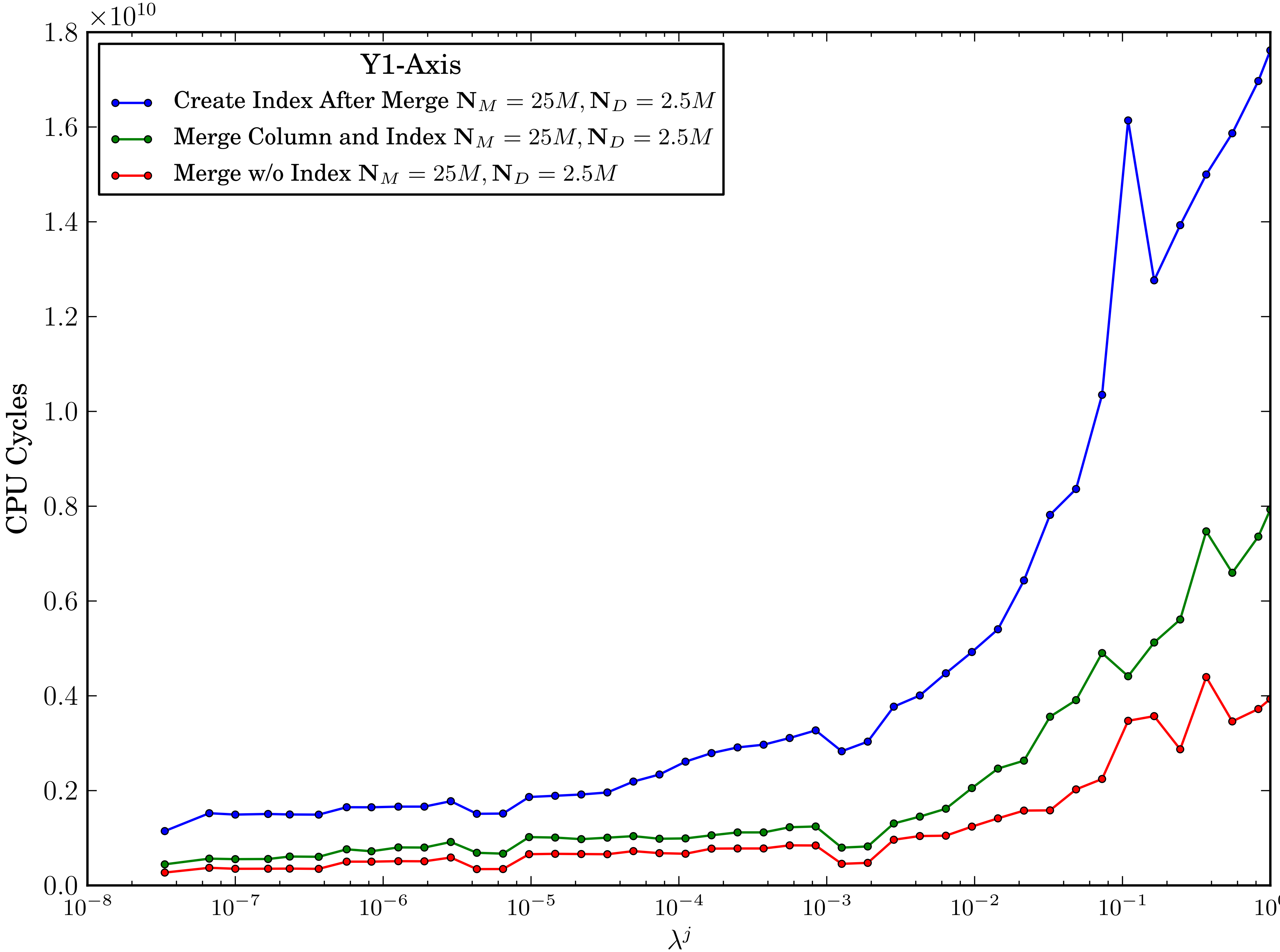
Index Viability



Distinct Value Fraction

$$\phi^j \approx \frac{MT_{ExtraRead} + MT_{ExtraWrite}}{MT_S}$$

Evaluation



Distinct Value Fraction

Conclusions

- Real-world measured CPU cycles are not only influenced by the memory traffic
  - Increased code size
  - Usage of write-combining buffers
- Memory traffic analysis allows to compare algorithms theoretically
- The general trend can be deducted from the analysis

Thank you!

Martin Faust
martin.faust@hpi.uni-potsdam.de