Hash tables

William Fiset

Outline

- What is a **Hash table(HT)** and what is a **hash function**?
- Properties of hash functions
- Discussion on collision resolution methods, in particular: **separate chaining** and **open addressing**
- Complexity analysis
- Separate chaining implementation details:
  - Linked list approach overview
  - Separate chaining FAQs
  - Separate chaining source code
- Separating chaining HT source code :)

Outline

- Open addressing techniques implementation details:
  - **Linear probing**
    - What is linear probing?
    - Chaos with cycles
    - Linear probing insertion examples
    - Table resizing and updating values
  - **Quadratic probing**
    - What is quadratic probing?
    - Problems with probing sequence cycles
    - Different ways to quadratically probe
    - Inserting/resize examples

Outline

- **Double hashing**
  - What is double hashing? How does it work?
  - Chaos with cycles
  - Constructing a new hash function
    - Universal hash functions
  - Inserting/resize example
- Removing elements
  - Solution using tombstones
  - Lazy deletion/relocation
  - Lots of examples
- Source code!

What is a Hash table?

A **Hash table (HT)** is a data structure that provides a mapping from keys to values using a technique called **hashing**.

What is a Hash table?

A **Hash table (HT)** is a data structure that provides a mapping from keys to values using a technique called **hashing**.

Key (name)	**Value (fav color)**
“William”	“green”
“Micah”	“purple”
“Catherine”	“yellow”
“Thomas”	“red”
“Leah”	“purple”

We refer to these as **key-value pairs**

Keys must be unique, but values can be repeated

What is a Hash table?

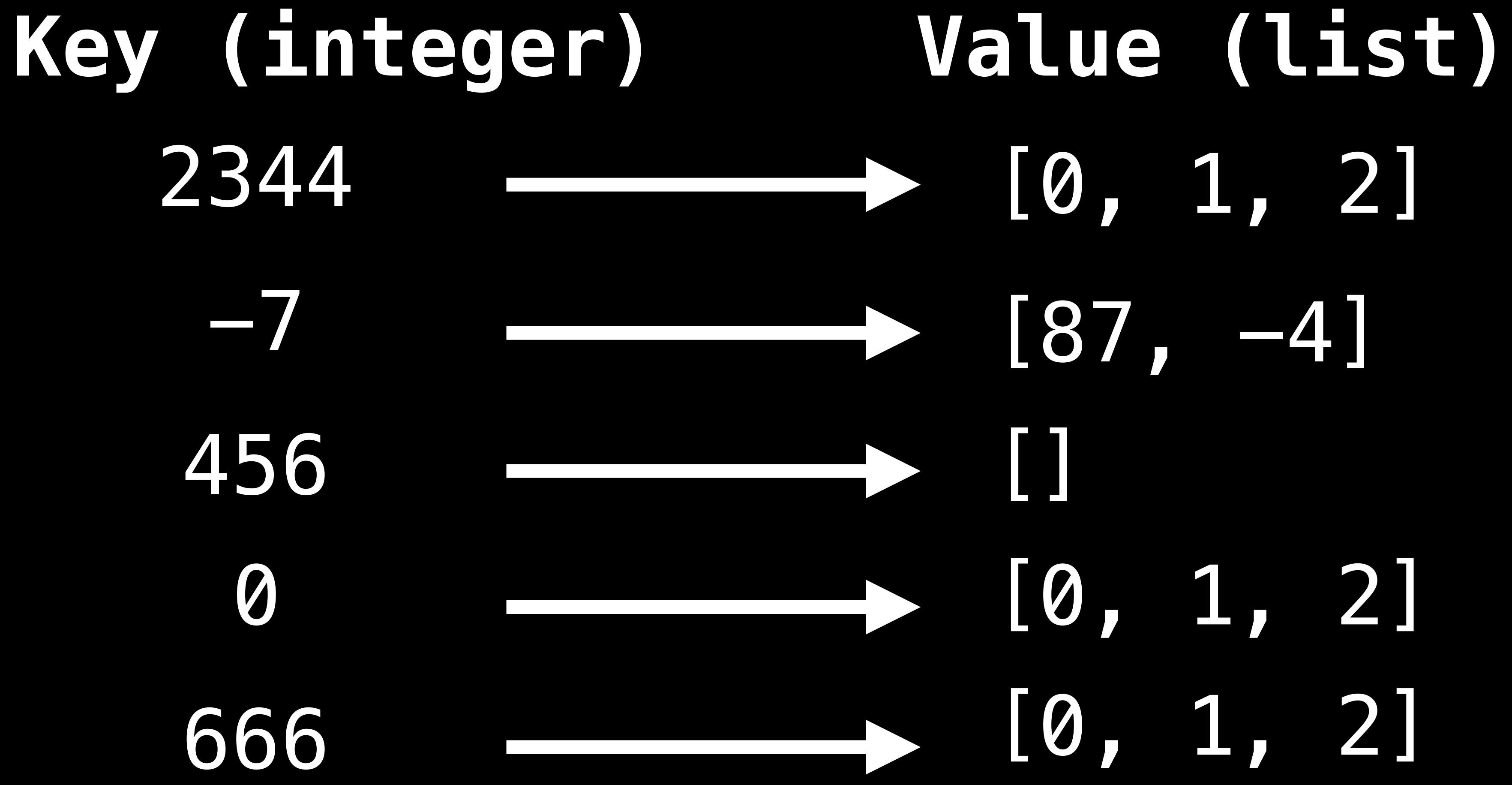
HTs are often used to track item frequencies. For instance, counting the number of times a word appears in a given text.

I parsed Shakespeare’s Hamlet (ignoring case and punctuation) and obtained the following frequency table:

Key (word)	Value (word frequency)
“hamlet”	114
"ghost"	33
“the"	1151
“lord"	223
“a"	550
“cabbage”	null
...	...

What is a Hash table?

The key-value pairs you can place in a HT can be of any type not just strings and numbers, but also objects! However, the keys needs to be **hashable**, a property we will discuss shortly.



Key (integer)	Value (list)
2344	[0, 1, 2]
-7	[87, -4]
456	[]
0	[0, 1, 2]
666	[0, 1, 2]

To be able to understand how a mapping is constructed between key–value pairs we first need to talk about hash functions.

To be able to understand how a mapping is constructed between key-value pairs we first need to talk about hash functions.

A **hash function** **H**(x) is a function that maps a key ‘x’ to a whole number in a fixed range.

To be able to understand how a mapping is constructed between key–value pairs we first need to talk about hash functions.

A **hash function** **H**(x) is a function that maps a key ‘x’ to a whole number in a fixed range.

For example, $\mathbf{H}(x) = (x^2 - 6x + 9) \bmod 10$ maps all integer keys to the range [0,9]

$$\begin{aligned}
\mathbf{H}(4) &= (16 - 24 + 9) \bmod 10 = 1 \\
\mathbf{H}(-7) &= (49 + 42 + 9) \bmod 10 = 0 \\
\mathbf{H}(0) &= (0 - 0 + 9) \bmod 10 = 9 \\
\mathbf{H}(2) &= (4 - 12 + 9) \bmod 10 = 1 \\
\mathbf{H}(8) &= (64 - 48 + 9) \bmod 10 = 5
\end{aligned}$$

We can also define hash functions for arbitrary objects such as strings, lists, tuples, multi data objects, etc…

We can also define hash functions for arbitrary objects such as strings, lists, tuples, multi data objects, etc…

For a string s let **H**(s) be a hash function defined below where **ASCII**(x) returns the ASCII value of the character x

```
ASCII(‘A’) = 65
ASCII(‘B’) = 66
      …
ASCII(‘Z’) = 90
```

For more check out
www.asciitable.com

```
function H(s):
    sum := 0
    for char in s:
        sum = sum + ASCII(char)
    return sum mod 50
```

We can also define hash functions for arbitrary objects such as strings, lists, tuples, multi data objects, etc…

For a string s let **H**(s) be a hash function defined below where **ASCII**(x) returns the ASCII value of the character x

ASCII(‘A’) = 65
ASCII(‘B’) = 66
…
ASCII(‘Z’) = 90

```
function H(s):
    sum := 0
    for char in s:
        sum = sum + ASCII(char)
    return sum mod 50
```

For more check out
www.asciitable.com

```
H(“BB”)  =       (66 + 66) mod 50 = 32
H(“”)    =             (0) mod 50 = 0
H(“ABC”) = (65 + 66 + 67) mod 50 = 48
H(“Z”)   =            (90) mod 50 = 40
```

Challenge: Suppose we have a database of people objects with three fields: name, age and sex. Can you define a hash function H(person) that maps a person to the set {0,1,2,3,4,5}?

Name	Age	Sex	Hash
William	21	M	?
Kate	19	F	?
Bob	33	M	?
Rose	26	F	?

Name	Age	Sex	Hash
William	21	M	?
Kate	19	F	?
Bob	33	M	?
Rose	26	F	?

There are an infinite number of possible valid hash functions H(person), here is one:

```
function H(person):
    hash := person.age
    hash = hash + length(person.name)
    if person.sex == “M”:
        hash = hash + 1
    return hash mod 6
```

Name	Age	Sex	Hash
William	21	M	5
Kate	19	F	5
Bob	33	M	1
Rose	26	F	0

There are an infinite number of possible valid hash functions H(person), here is one:

```
function H(person):
    hash := person.age
    hash = hash + length(person.name)
    if person.sex == “M”:
        hash = hash + 1
    return hash mod 6
```

Properties of Hash functions

If **H(x) = H(y)** then objects x and y **might be equal**, but if **H(x) ≠ H(y)** then x and y are **certainly not equal**.

Properties of Hash functions

If **H(x) = H(y)** then objects x and y **might be equal**, but if **H(x) ≠ H(y)** then x and y are **certainly not equal**.

Q: How can we use this to our advantage to speedup object comparisons?

Properties of Hash functions

If **H(x) = H(y)** then objects x and y **might be equal**, but if **H(x) ≠ H(y)** then x and y are **certainly not equal**.

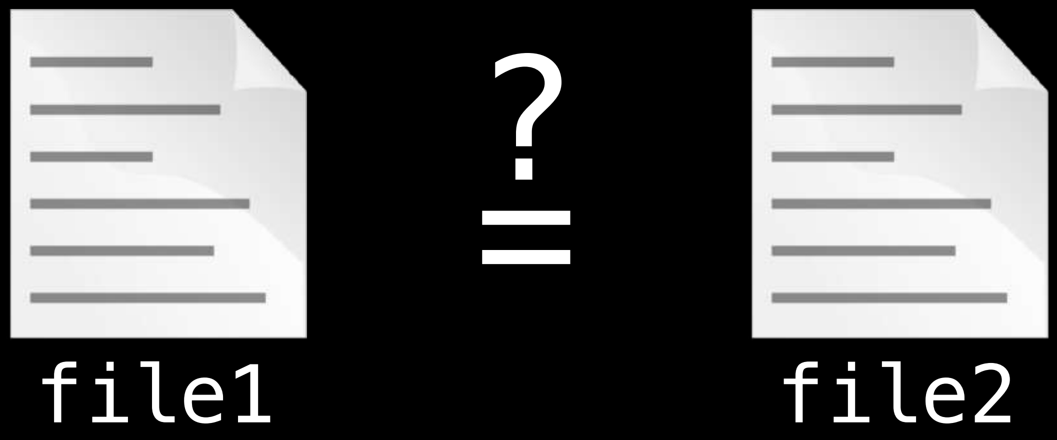
Q: How can we use this to our advantage to speedup object comparisons?

A: This means that instead of comparing x and y directly a smarter approach is to first compare their hash values, and only if the hash values match do we need to explicitly compare x and y.

Properties of Hash functions

Consider the problem of trying to determine if two very large files have the same contents.

If we precomputed H(file1) and H(file2) first we should compare those hash values since comparing hash values is **O(1)**! If possible, we do not want to open either of the files directly. Comparing their contents can be very slow, although we may have to if H(file1) = H(file2).



NOTE: Hash functions for files are more sophisticated than those used for hashtables. Instead for files we use what are called **cryptographic hash functions** also called **checksums**.

Properties of Hash functions

A hash function **H**(x) must be **deterministic**.

This means that if **H**(x) = y then **H**(x) must always produce y and never another value. This may seen obvious, but it is critical to the functionality of a hash function.

Properties of Hash functions

A hash function **H**(x) must be **deterministic**.

This means that if **H**(x) = y then **H**(x) must always produce y and never another value. This may seen obvious, but it is critical to the functionality of a hash function.

Example of non-deterministic hash function:

```
counter := 0
function H(x):
    counter = counter + 1
    return (x + counter) mod 13
```

The first time called **H**(5) = 6, but if called again **H**(5) = 7!

Properties of Hash functions

We try very hard to make **uniform** hash functions to minimize the number of hash collisions.

A **hash collision** is when two objects x, y hash to the same value (i.e. **H**(x) = **H**(y)).

Properties of Hash functions

We try very hard to make **uniform** hash functions to minimize the number of hash collisions.

A **hash collision** is when two objects x, y hash to the same value (i.e. **H**(x) = **H**(y)).

In the table we generated earlier William and Kate have a hash collision.

Name	Age	Sex	Hash
William	21	M	**5**
Kate	19	F	**5**
Bob	33	M	**1**
Rose	26	F	**0**

We are now able to answer a central question about the types of keys we are allowed to use in our hashtable:

Q: What makes a key of type T **hashable** ?

We are now able to answer a central question about the types of keys we are allowed to use in our hashtable:

Q: What makes a key of type T **hashable** ?

A: Since we are going to use hash functions in the implementation of our hash table we need our hash functions to be deterministic. To enforce this behaviour, we demand that the **keys used in our hash table are immutable** data types. Hence, if a key of type T is immutable, and we have a hash function **H**(k) defined for all keys k of type T then we say a key of type T is hashable.

How does a hash table work?

Ideally we would like to have a very fast insertion, lookup and removal time for the data we are placing within our hash table.

Remarkably, we can achieve all this in **O(1)*** time using a **hash function as a way to index into a hash table**.

* The constant time behaviour attributed to hash tables is only true if you have a good **uniform hash function**!

How does a hash table work?

Think of the hash table on the right as an indexable block of memory (an array) and we can only access its entries using the value given to us by our hash function H(x)

	Key	Value
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		

How does a hash table work?

Suppose we're inserting (integer, string) key-value pairs into the table representing rankings of users to their usernames from an online programming competition and we're using the hash function:

$$H(x) = x^2 + 3 \bmod 10$$

	Key	Value
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		

How does a hash table work?

Suppose we’re inserting (integer, string) key-value pairs into the table representing rankings of users to their usernames from an online programming competition and we’re using the hash function:

$$H(x) = x^2 + 3 \bmod 10$$

To **insert** (3, “byte-eater”) we hash the key (the rank) and find out where it goes in the table

$$H(3) = (3^2 + 3) \bmod 10 = 2$$

	Key	Value
0		
1		
2	3	“byte-eater”
3		
4		
5		
6		
7		
8		
9		

How does a hash table work?

Suppose we're inserting (integer, string) key-value pairs into the table representing rankings of users to their usernames from an online programming competition and we're using the hash function:

$$H(x) = x^2 + 3 \bmod 10$$

To **insert** (1, "will.fiset") we hash the key (the rank) and find out where it goes in the table

$$H(1) = (1^2 + 3) \bmod 10 = 4$$

	Key	Value
0		
1		
2	3	"byte-eater"
3		
4	1	"will.fiset"
5		
6		
7		
8		
9		

How does a hash table work?

Suppose we're inserting (integer, string) key-value pairs into the table representing rankings of users to their usernames from an online programming competition and we're using the hash function:

$$H(x) = x^2 + 3 \bmod 10$$

To **insert** (32, “Lauren425”) we hash the key (the rank) and find out where it goes in the table

$$H(1) = (32^2 + 3) \bmod 10 = 7$$

	Key	Value
0		
1		
2	3	“byte-eater”
3		
4	1	“will.fiset”
5		
6		
7	32	“Lauren425”
8		
9		

How does a hash table work?

Suppose we're inserting (integer, string) key-value pairs into the table representing rankings of users to their usernames from an online programming competition and we're using the hash function:

$$H(x) = x^2 + 3 \bmod 10$$

To **insert** (5, “ternarywizard”) we hash the key (the rank) and find out where it goes in the table

$$H(5) = (5^2 + 3) \bmod 10 = 8 \rightarrow$$

	Key	Value
0		
1		
2	3	“byte-eater”
3		
4	1	“will.fiset”
5		
6		
7	32	“Lauren425”
8	5	“ternarywizard"
9		

How does a hash table work?

Suppose we're inserting (integer, string) key-value pairs into the table representing rankings of users to their usernames from an online programming competition and we're using the hash function:

$$H(x) = x^2 + 3 \bmod 10$$

To **insert** (10, “orange-knight”) we hash the key (the rank) and find out where it goes in the table

$$H(10) = (10^2 + 3) \bmod 10 = 3$$

	Key	Value
0		
1		
2	3	“byte-eater”
3	10	“orange-knight”
4	1	“will.fiset”
5		
6		
7	32	“Lauren425”
8	5	“ternarywizard"
9		

How does a hash table work?

Suppose we're inserting (integer, string) key-value pairs into the table representing rankings of users to their usernames from an online programming competition and we're using the hash function:

$$H(x) = x^2 + 3 \bmod 10$$

To **lookup** which user has rank r we simply compute $H(r)$ and look inside the hashtable!

	Key	Value
0		
1		
2	3	"byte-eater"
3	10	"orange-knight"
4	1	"will.fiset"
5		
6		
7	32	"Lauren425"
8	5	"ternarywizard"
9		

How does a hash table work?

Q: What do we do if there is a **hash collision**?

For example, users with ranks 2 and 8 hash to the same value!!

H(2) = 2^2+3 mod 10 = 7 = 8^2+3 mod 10 = **H**(8)

How does a hash table work?

Q: What do we do if there is a **hash collision**?

For example, users with ranks 2 and 8 hash to the same value!!

$\mathbf{H}(2) = 2^2+3 \bmod 10 = 7 = 8^2+3 \bmod 10 = \mathbf{H}(8)$

A: We use one of many hash collision resolution techniques to handle this, the two most popular ones are **separate chaining** and **open addressing**.

How does a hash table work?

Separate chaining deals with hash collisions by maintaining a data structure (usually a linked list) to hold all the different values which hashed to a particular value.

Open addressing deals with hash collisions by finding another place within the hash table for the object to go by offsetting it from the position to which it hashed to.

Complexity

Operation	Average	Worst
Insertion	O(1)*	O(n)
Removal	O(1)*	O(n)
Search	O(1)*	O(n)

* The constant time behaviour attributed to hash tables is only true if you have a good **uniform hash function**!

Next Video: Separate chaining

Hash table implementation and source code and tests can all be found at the following link:

github.com/williamfiset/data-structures

Hash table Separate chaining

A quick look at the separate chaining collision resolution technique

William Fiset

What is Separate Chaining?

Separate chaining is one of many strategies to deal with hash collisions by maintaining a data structure (usually a linked list) to hold all the different values which hashed to a particular value.

What is Separate Chaining?

Separate chaining is one of many strategies to deal with hash collisions by maintaining a data structure (usually a linked list) to hold all the different values which hashed to a particular value.

NOTE: The data structure used to cache the items which hashed to a particular value is not limited to a linked list. Some implementations use one or a mixture of: arrays, binary trees, self balancing trees and etc…

Linked list Separate Chaining Insertion

Suppose we have a hash table that will store (name, age) key-value pairs and we wish to insert the following entries:

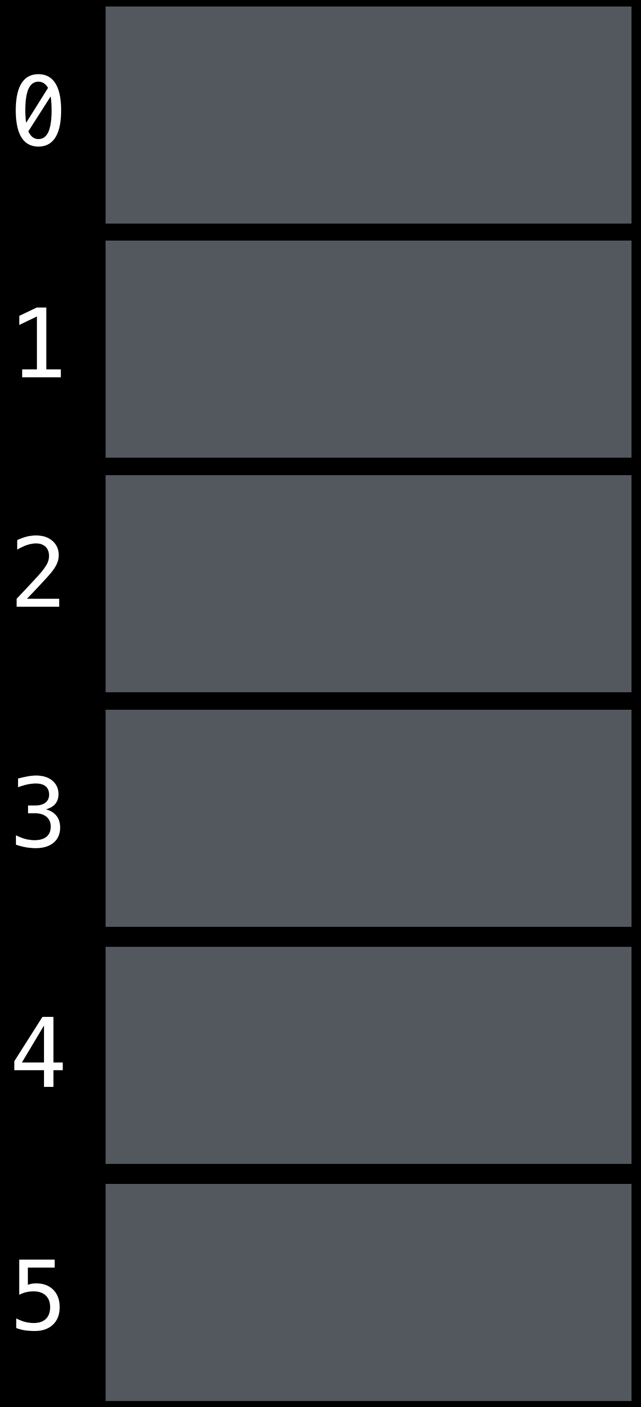
Name	Age	Hash
Will	21	
Leah	18	
Rick	61	
Rai	25	
Lara	34	
Ryan	56	
Lara	34	
Finn	21	
Mark	10	

Linked list Separate Chaining Insertion

Using an arbitrary hash function defined for strings we can assign each key a hash value.

Name	Age	Hash
Will	21	3
Leah	18	4
Rick	61	2
Rai	25	1
Lara	34	4
Ryan	56	1
Lara	34	4
Finn	21	3
Mark	10	4

Linked list Separate Chaining Insertion



0

1

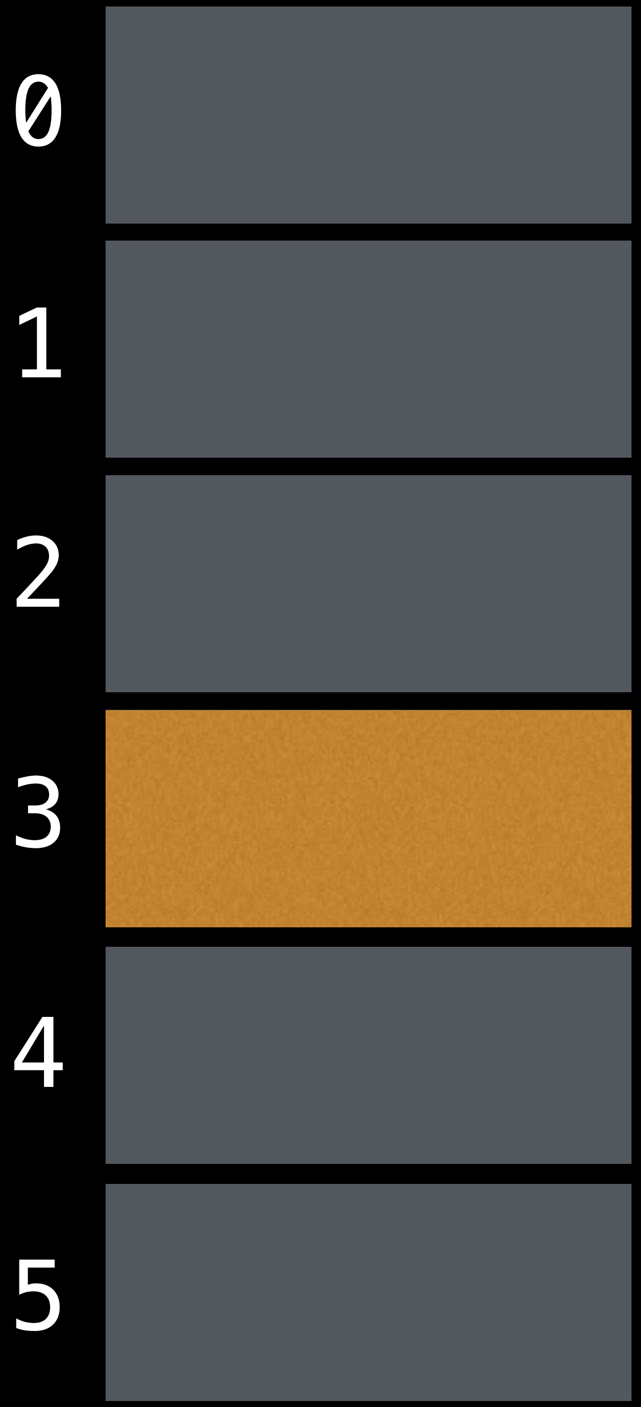
2

3

4

5

Linked list Separate Chaining Insertion

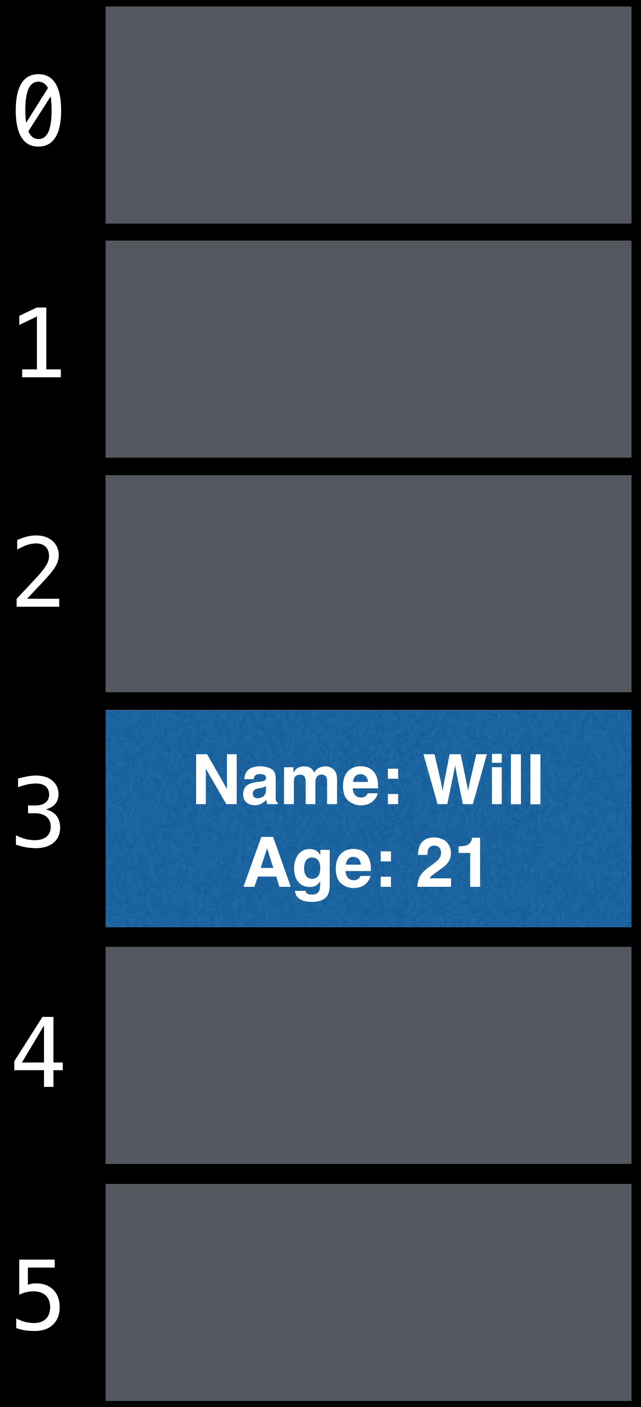


Name: Will

Age: 21

Hash: 3

Linked list Separate Chaining Insertion



0

1

2

3 Name: Will
Age: 21

4

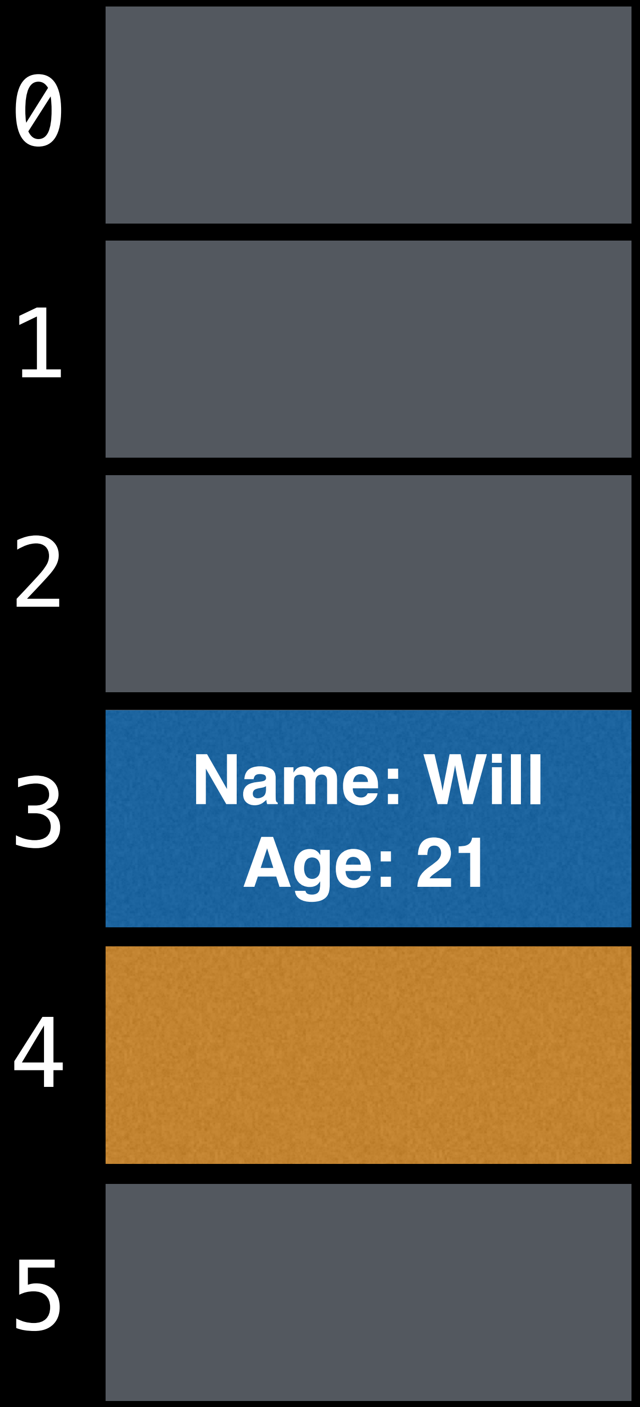
5

Name: Will

Age: 21

Hash: 3

Linked list Separate Chaining Insertion



0

1

2

3 Name: Will
Age: 21

4

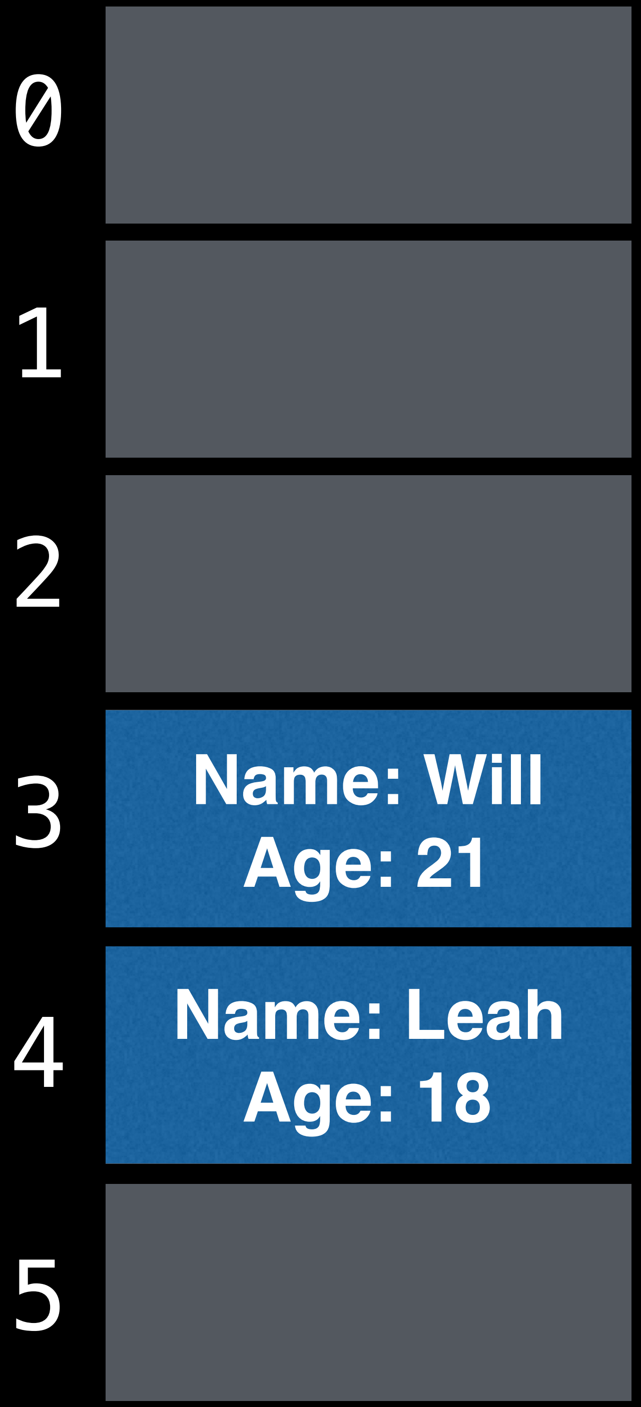
5

Name: Leah

Age: 18

Hash: 4

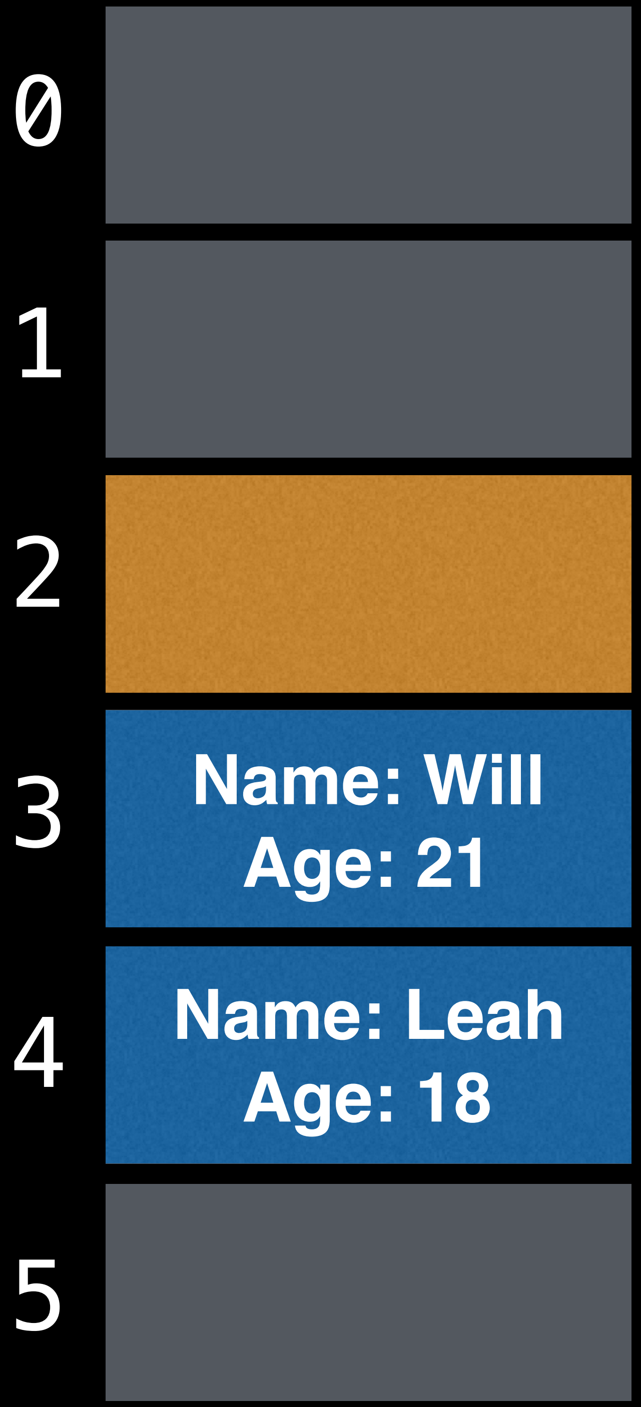
Linked list Separate Chaining Insertion



0	
1	
2	
3	Name: Will Age: 21
4	Name: Leah Age: 18
5	

Name: Leah
Age: 18
Hash: 4

Linked list Separate Chaining Insertion



0

1

2

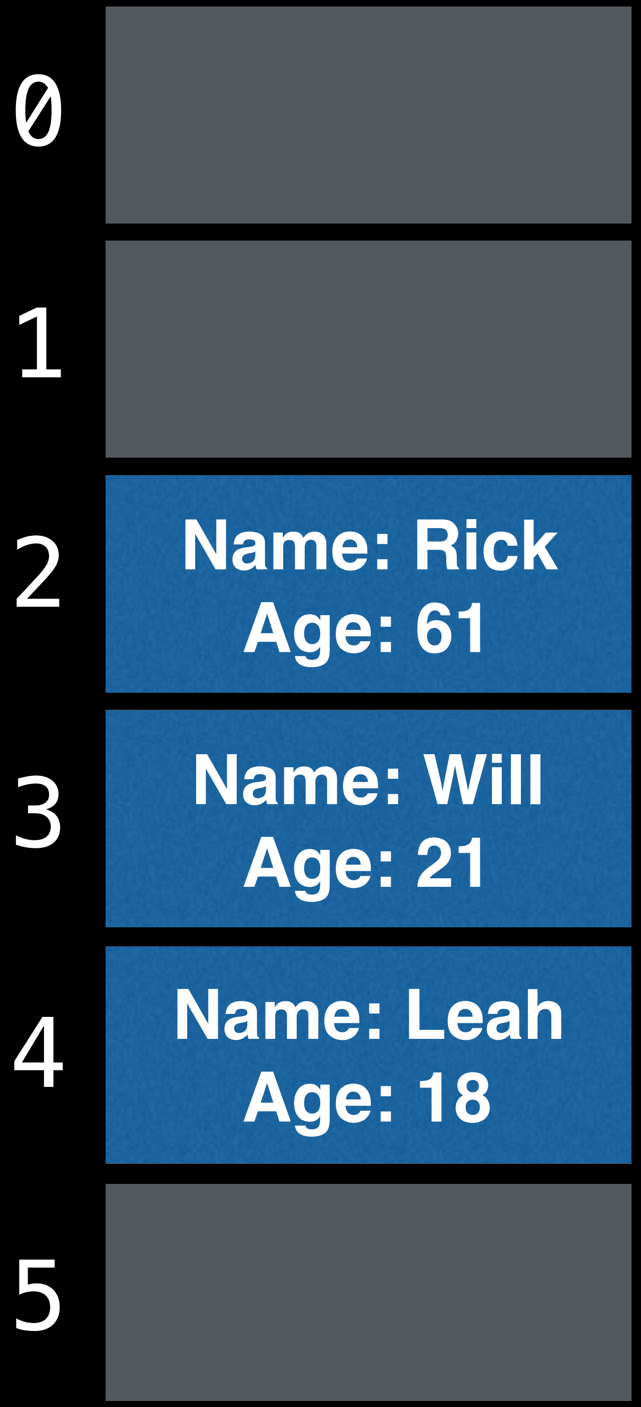
3 Name: Will
Age: 21

4 Name: Leah
Age: 18

5

Name: Rick
Age: 61
Hash: 2

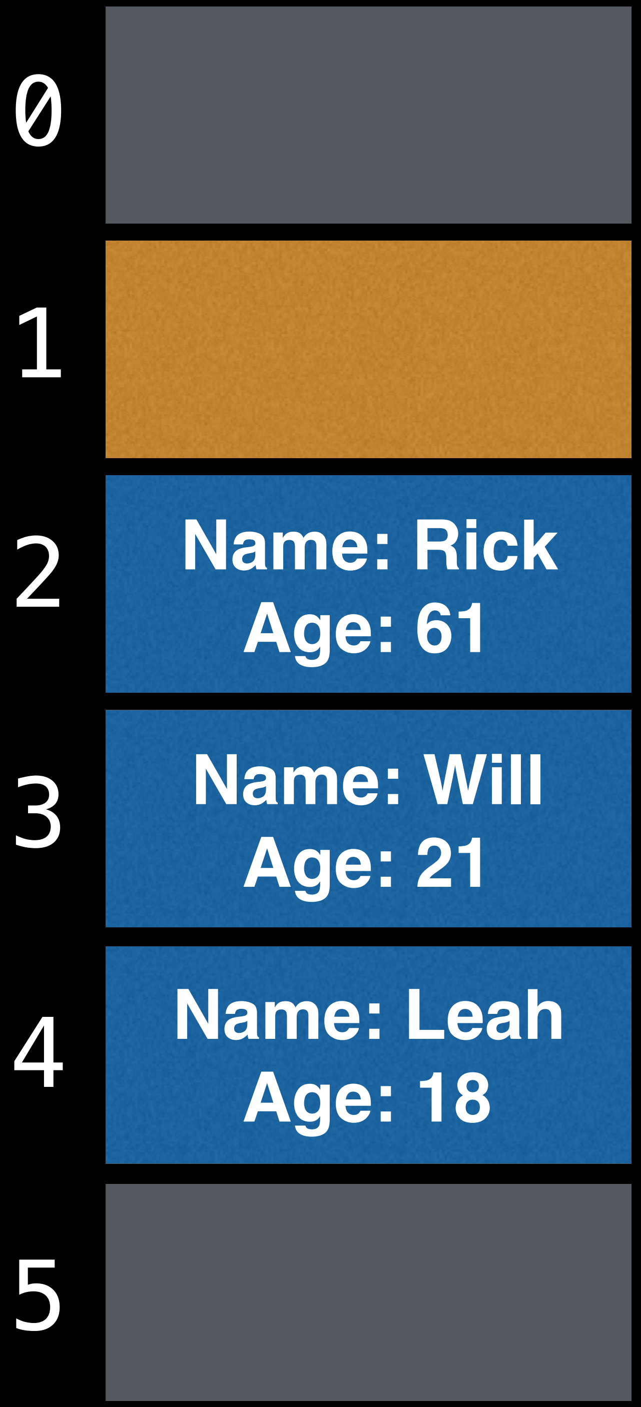
Linked list Separate Chaining Insertion



0	
1	
2	Name: Rick Age: 61
3	Name: Will Age: 21
4	Name: Leah Age: 18
5	

Name: Rick
Age: 61
Hash: 2

Linked list Separate Chaining Insertion



0

1

2 Name: Rick
Age: 61

3 Name: Will
Age: 21

4 Name: Leah
Age: 18

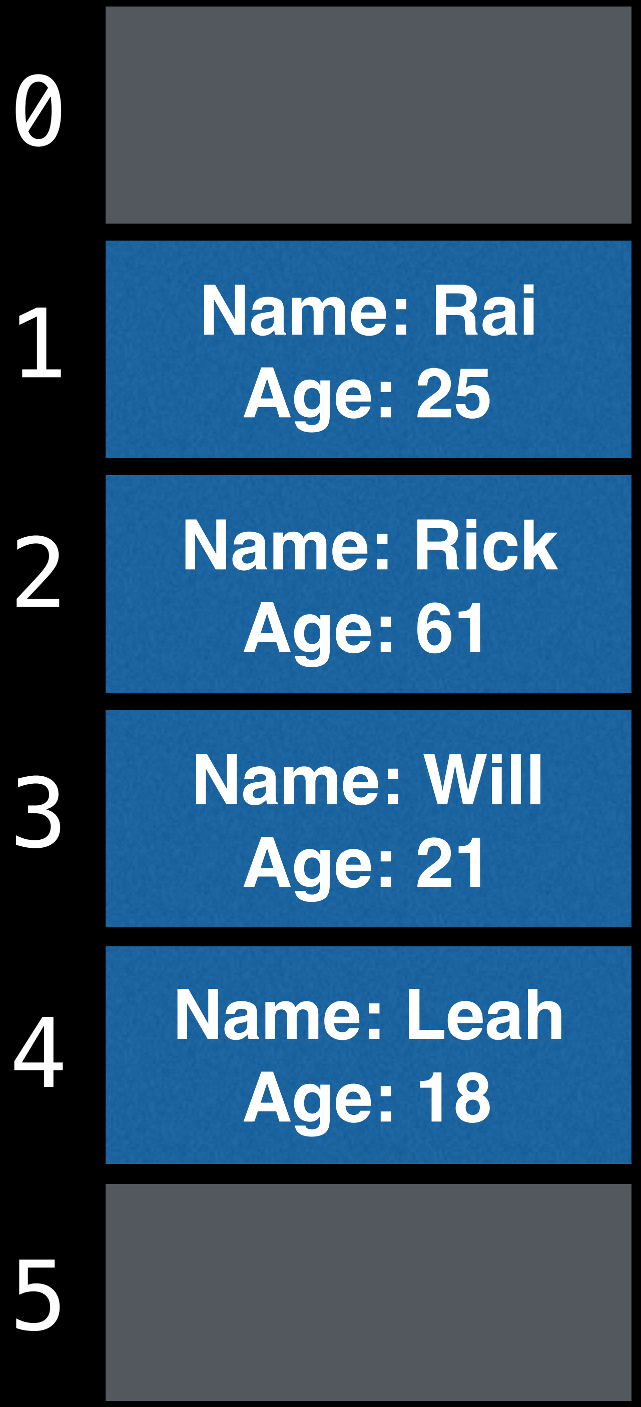
5

Name: Ria

Age: 25

Hash: 1

Linked list Separate Chaining Insertion



0

1 Name: Rai
Age: 25

2 Name: Rick
Age: 61

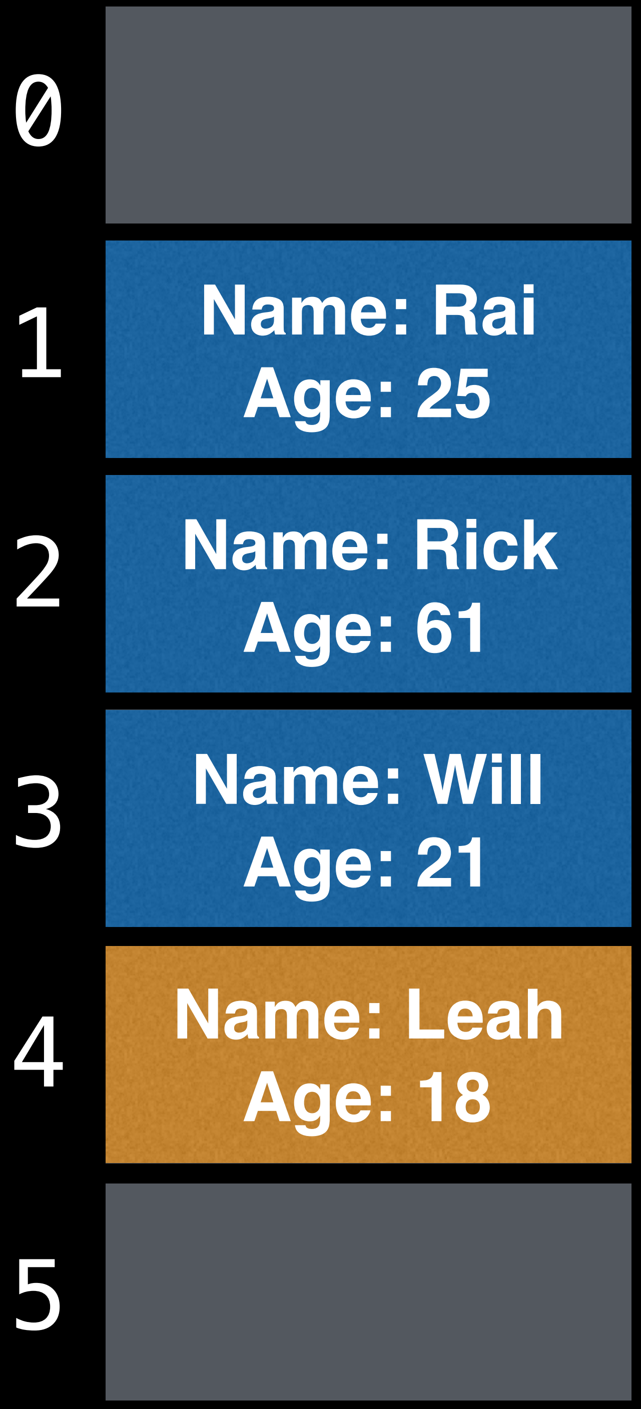
3 Name: Will
Age: 21

4 Name: Leah
Age: 18

5

Name: Ria
Age: 25
Hash: 1

Linked list Separate Chaining Insertion



0

1 Name: Rai Age: 25

2 Name: Rick Age: 61

3 Name: Will Age: 21

4 Name: Leah Age: 18

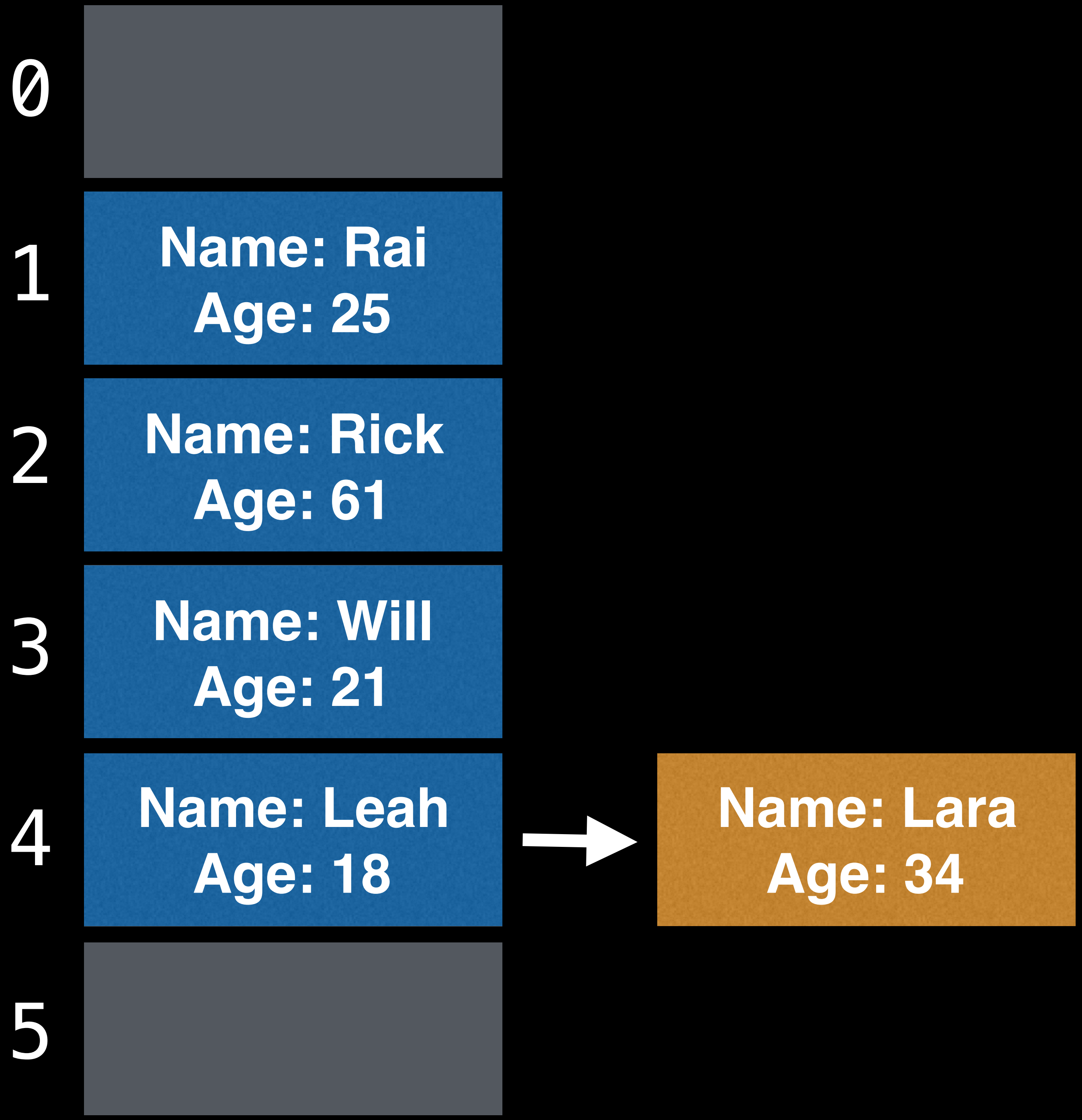
5

Name: Lara

Age: 34

Hash: 4

Linked list Separate Chaining Insertion



0

1 Name: Rai
Age: 25

2 Name: Rick
Age: 61

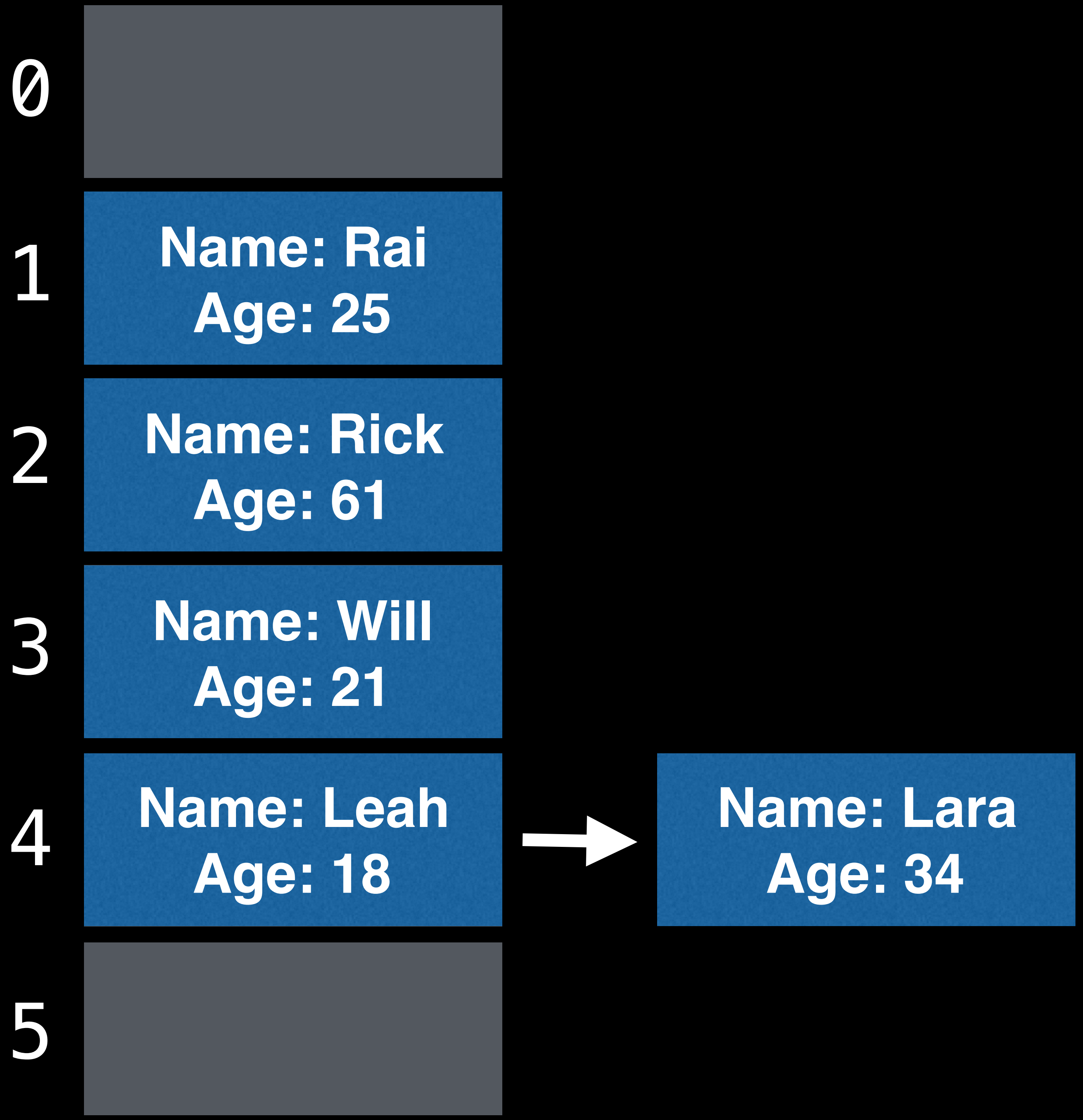
3 Name: Will
Age: 21

4 Name: Leah
Age: 18 → Name: Lara
Age: 34

5

Name: Lara
Age: 34
Hash: 4

Linked list Separate Chaining Insertion



0

1 Name: Rai Age: 25

2 Name: Rick Age: 61

3 Name: Will Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34

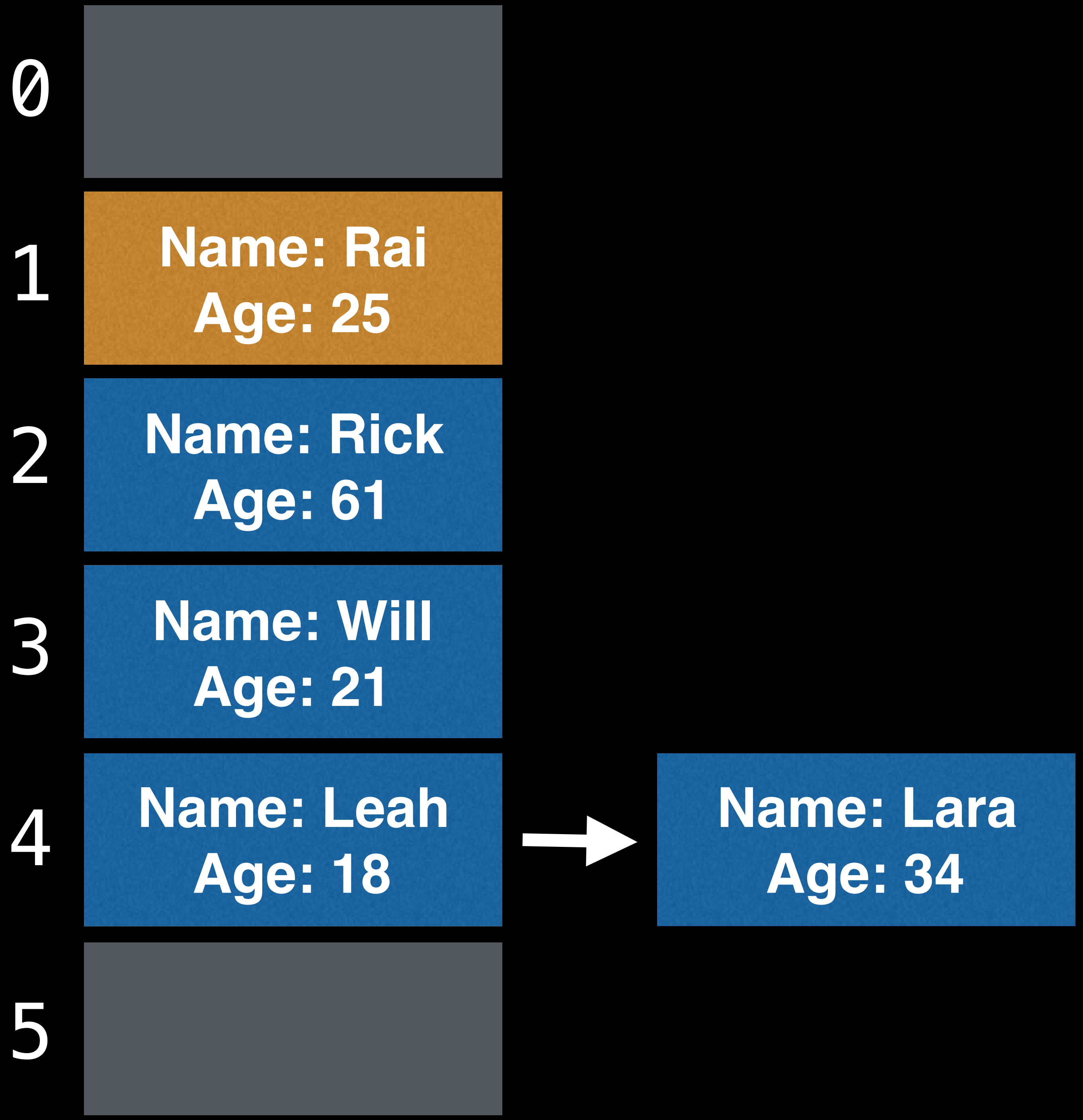
5

Name: Lara

Age: 34

Hash: 4

Linked list Separate Chaining Insertion



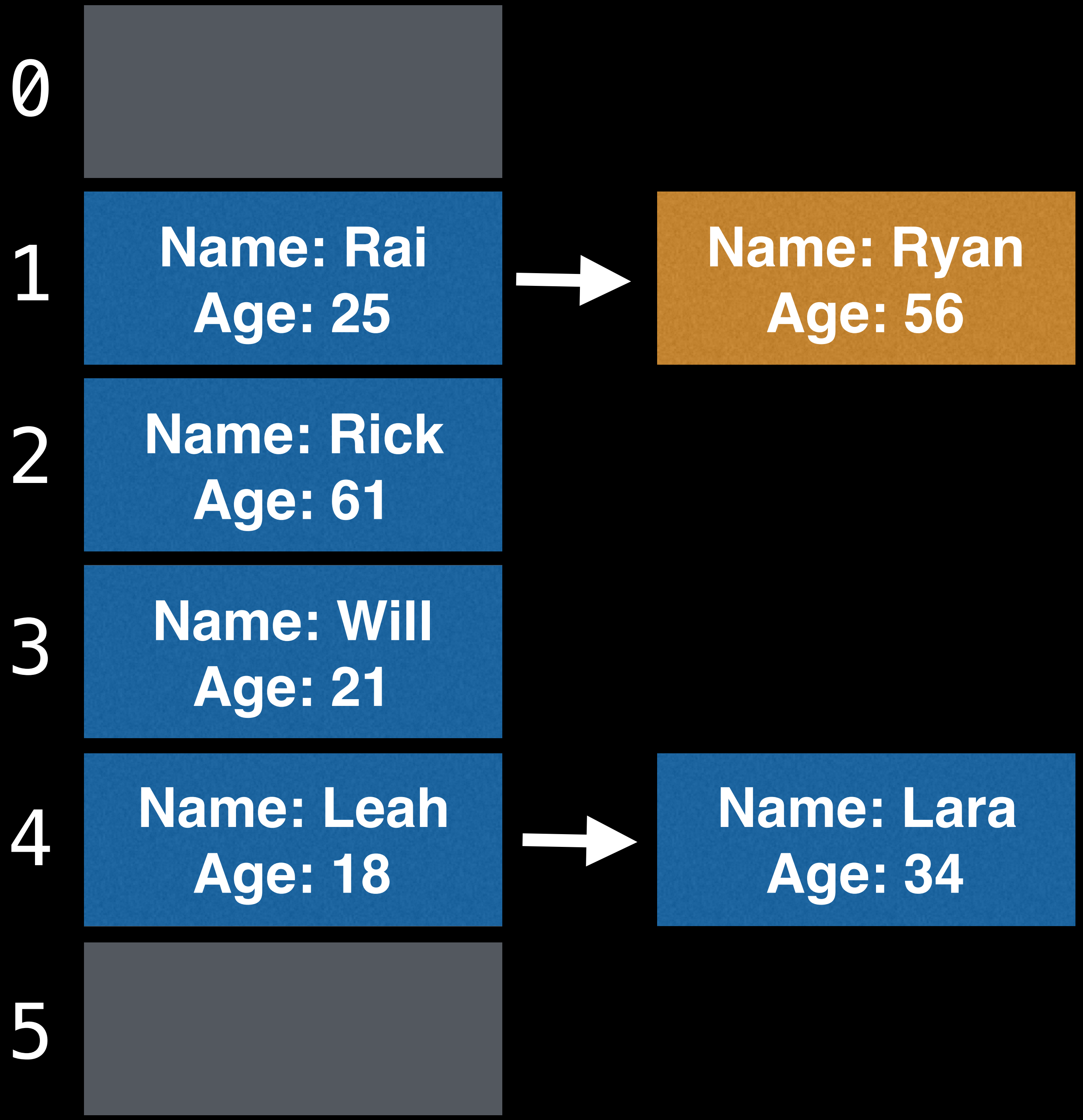
0		
1	Name: Rai Age: 25	
2	Name: Rick Age: 61	
3	Name: Will Age: 21	
4	Name: Leah Age: 18	Name: Lara Age: 34
5		

Name: Ryan

Age: 56

Hash: 1

Linked list Separate Chaining Insertion



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

3 Name: Will Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34

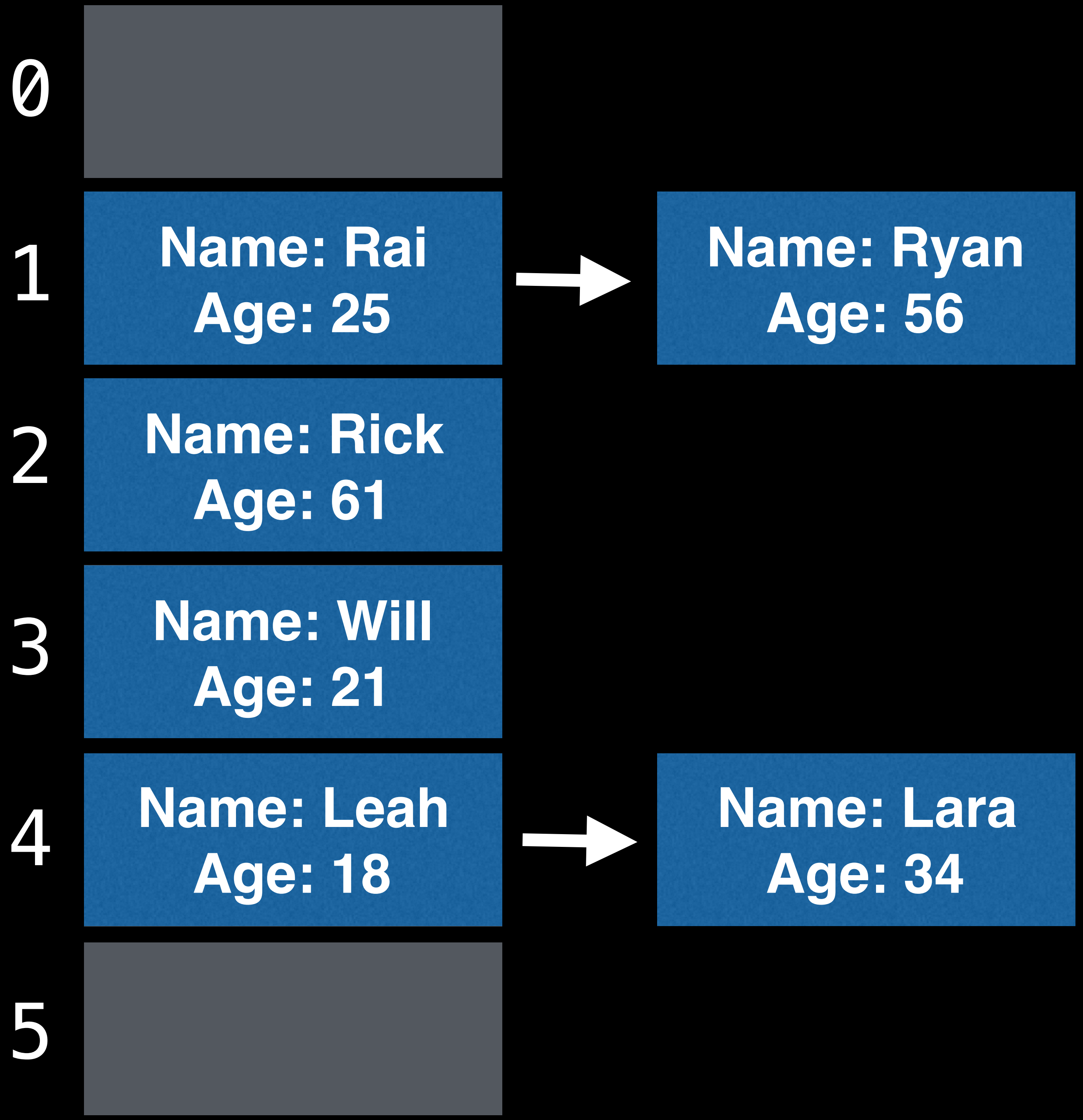
5

Name: Ryan

Age: 56

Hash: 1

Linked list Separate Chaining Insertion



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

3 Name: Will Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34

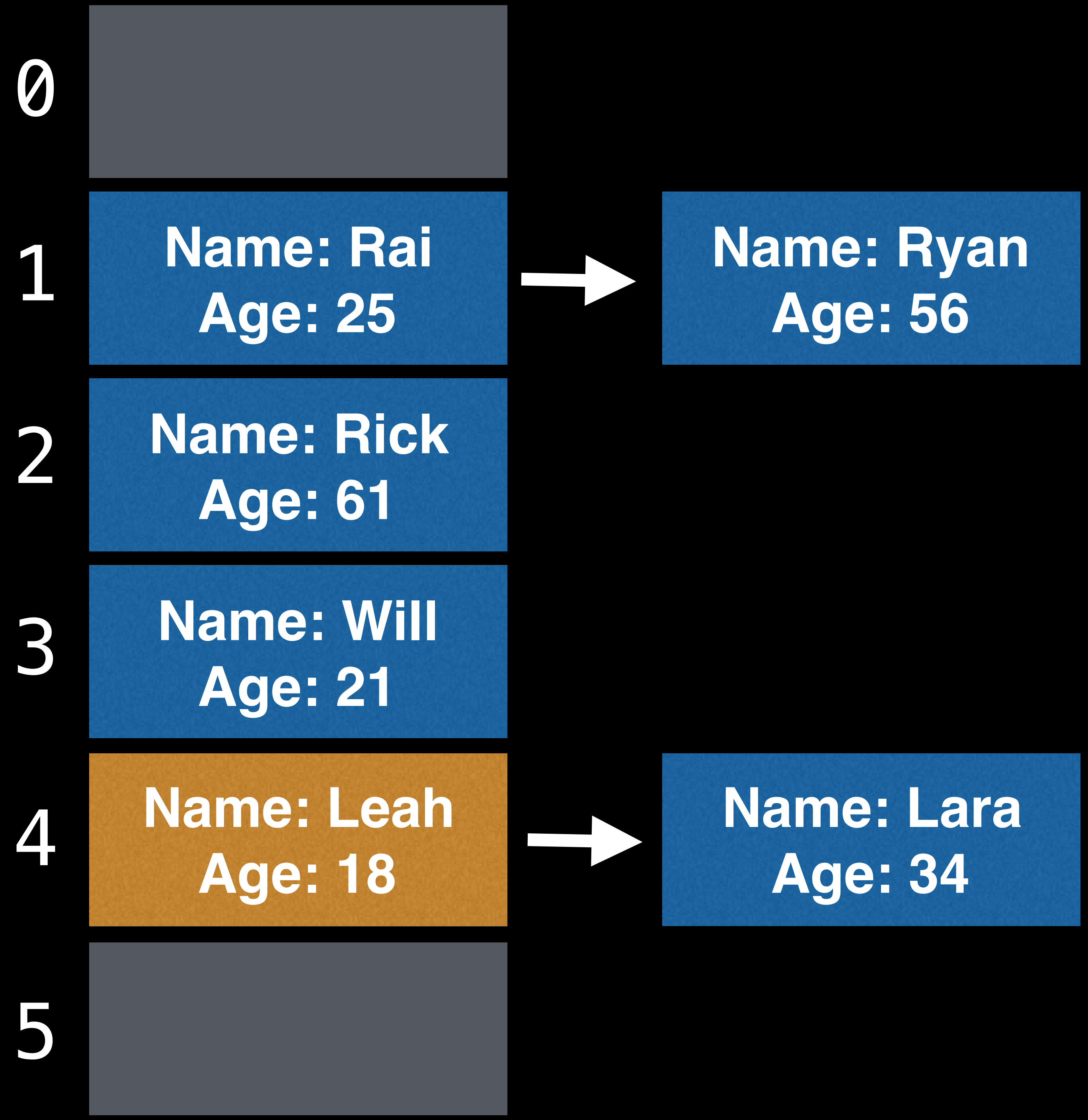
5

Name: Ryan

Age: 56

Hash: 1

Linked list Separate Chaining Insertion



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

3 Name: Will Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34

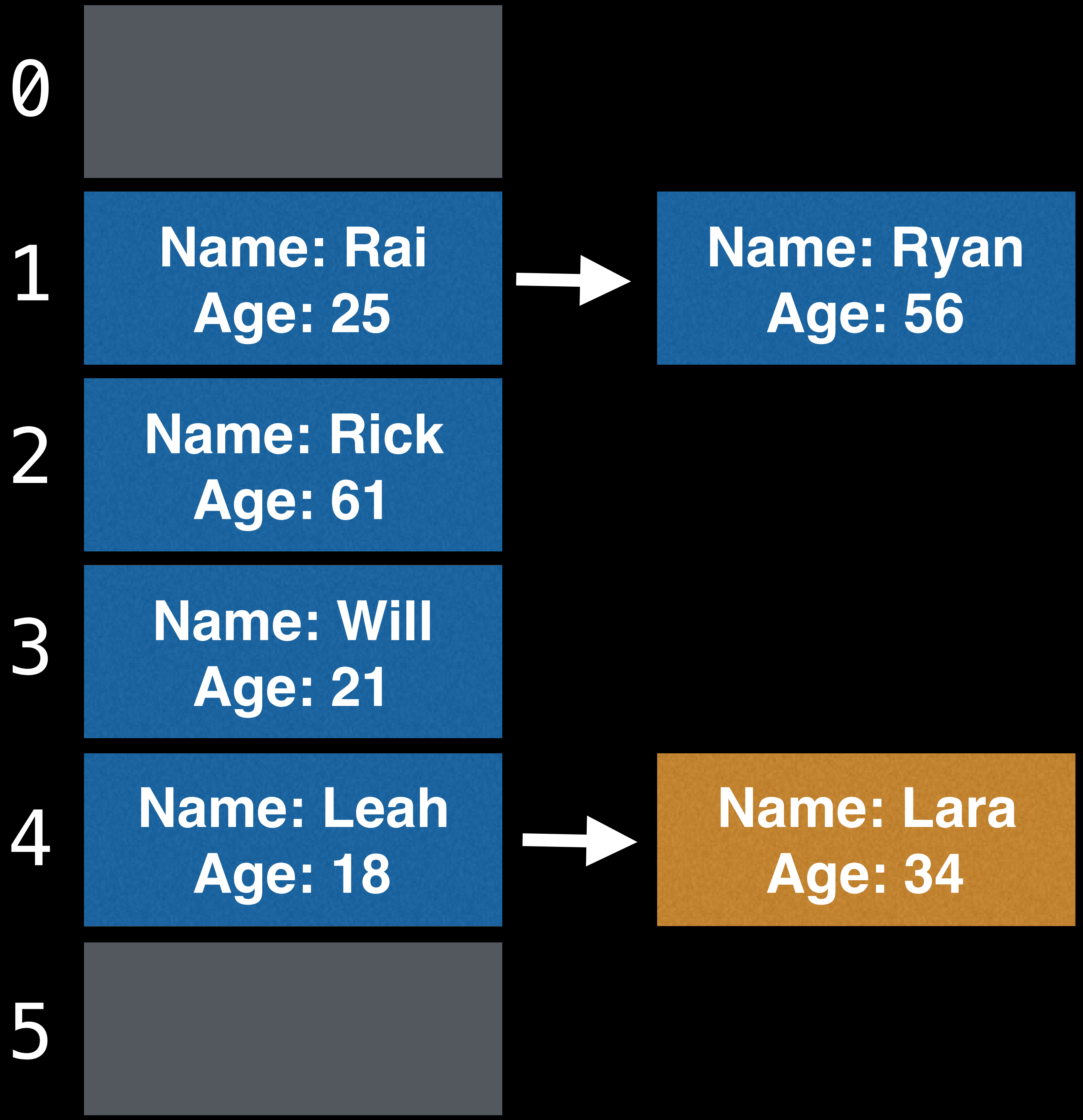
5

Name: Lara

Age: 34

Hash: 4

Linked list Separate Chaining Insertion



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

3 Name: Will Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34

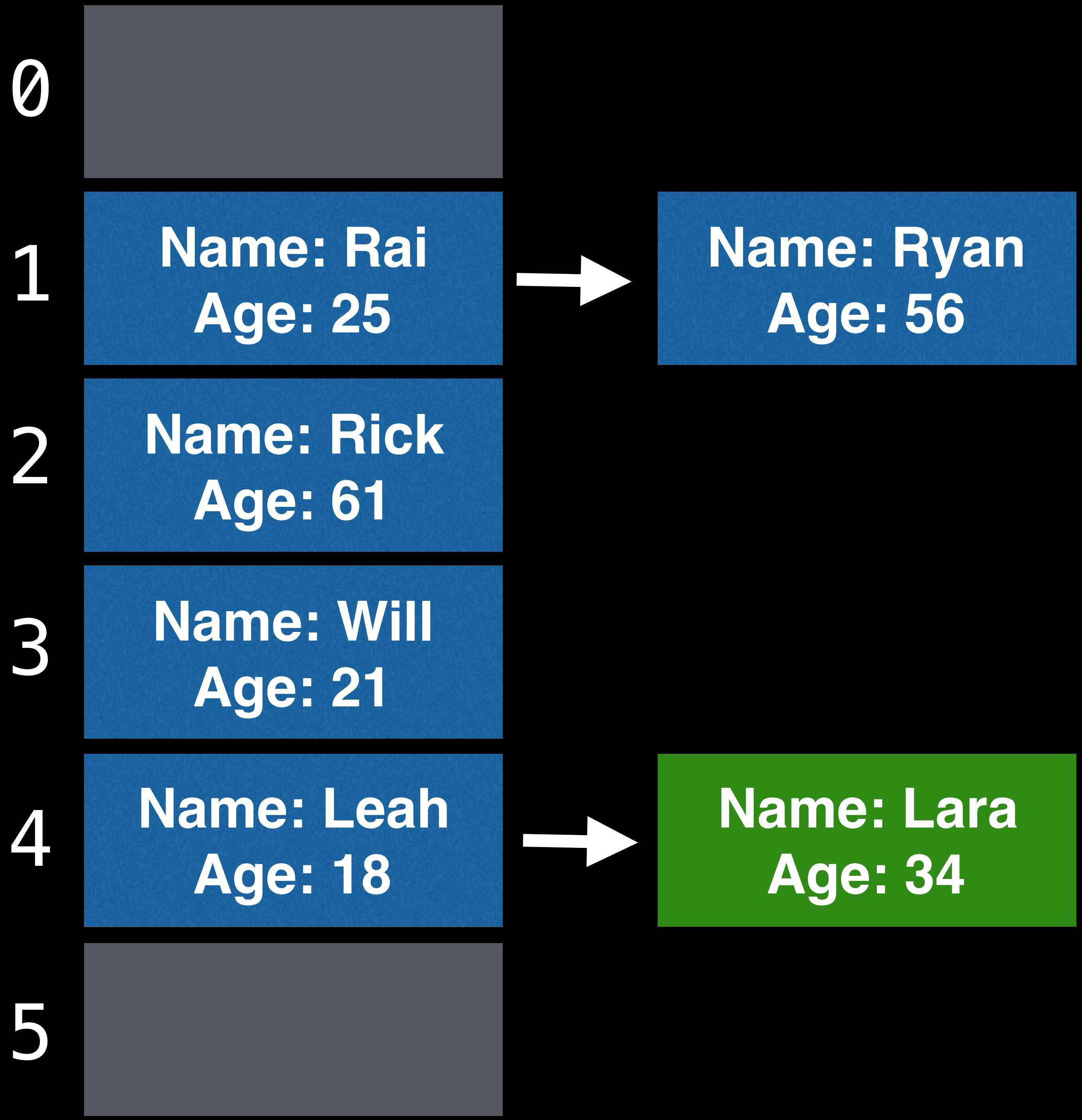
5

Name: Lara

Age: 34

Hash: 4

Linked list Separate Chaining Insertion



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

3 Name: Will Age: 21

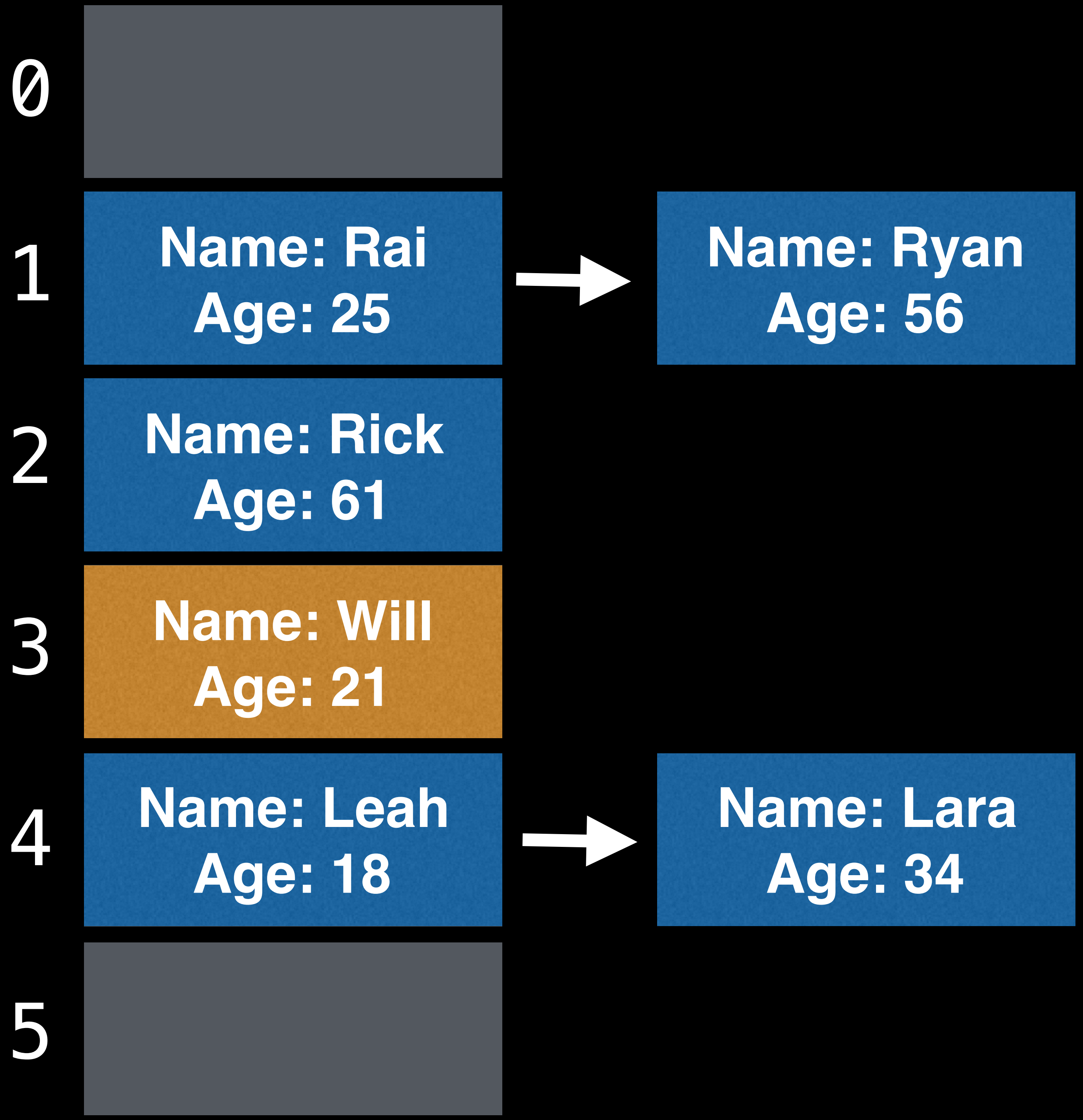
4 Name: Leah Age: 18 → Name: Lara Age: 34

5

Name: Lara
Age: 34
Hash: 4

Lara already exists in the hashtable!

Linked list Separate Chaining Insertion



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

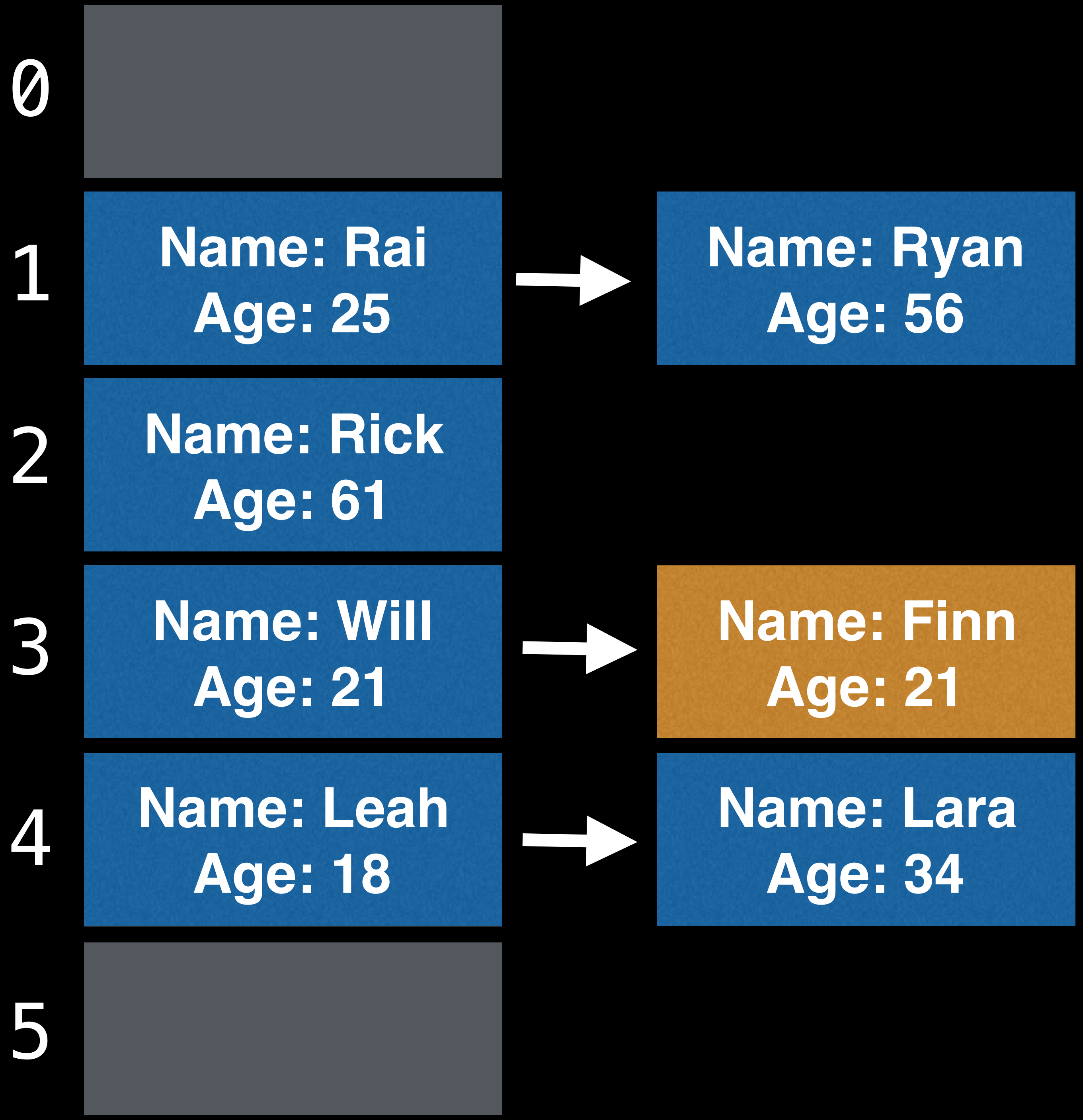
3 Name: Will Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34

5

Name: Finn
Age: 21
Hash: 3

Linked list Separate Chaining Insertion



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

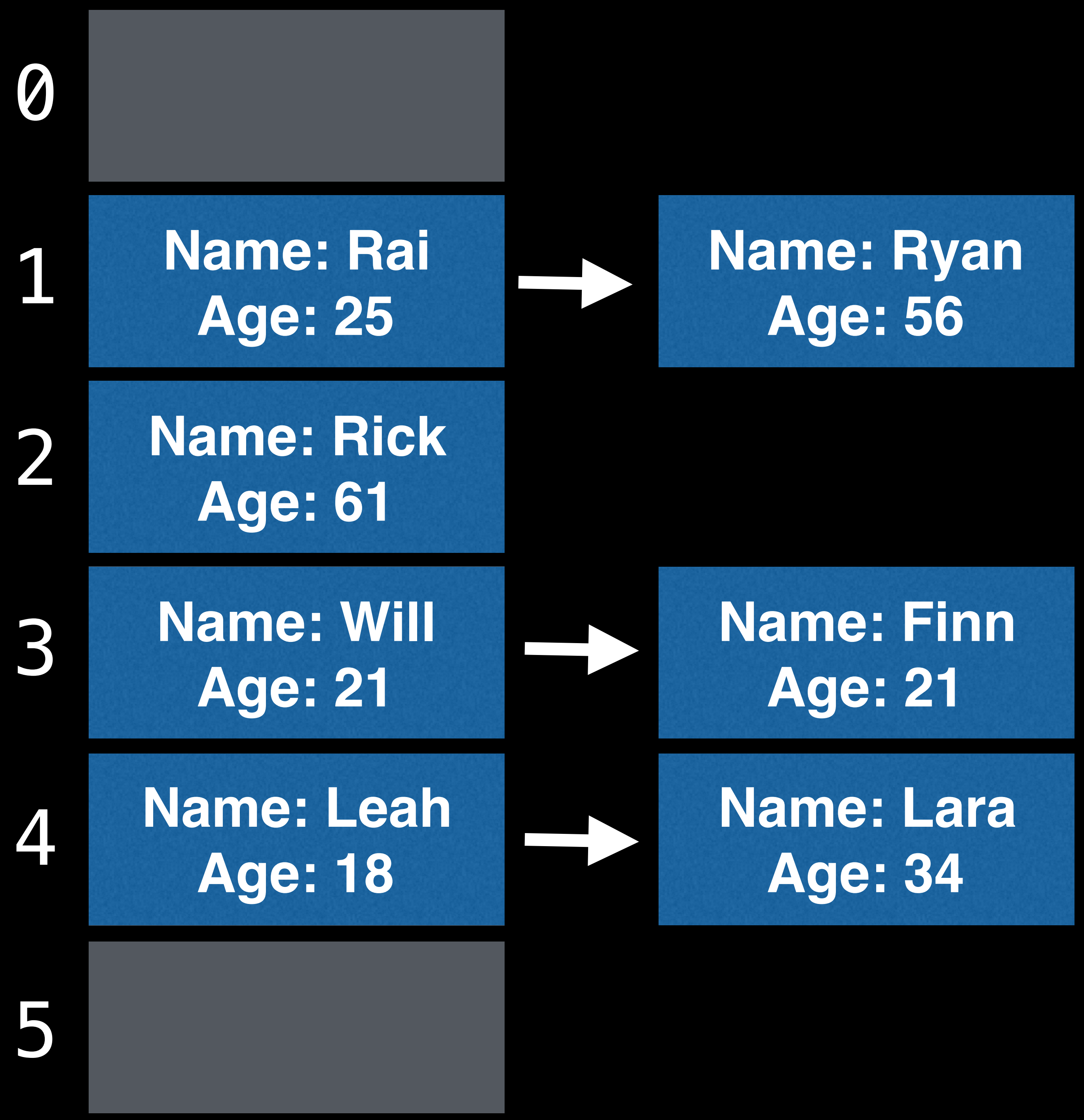
3 Name: Will Age: 21 → Name: Finn Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34

5

Name: Finn
Age: 21
Hash: 3

Linked list Separate Chaining Insertion



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

3 Name: Will Age: 21 → Name: Finn Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34

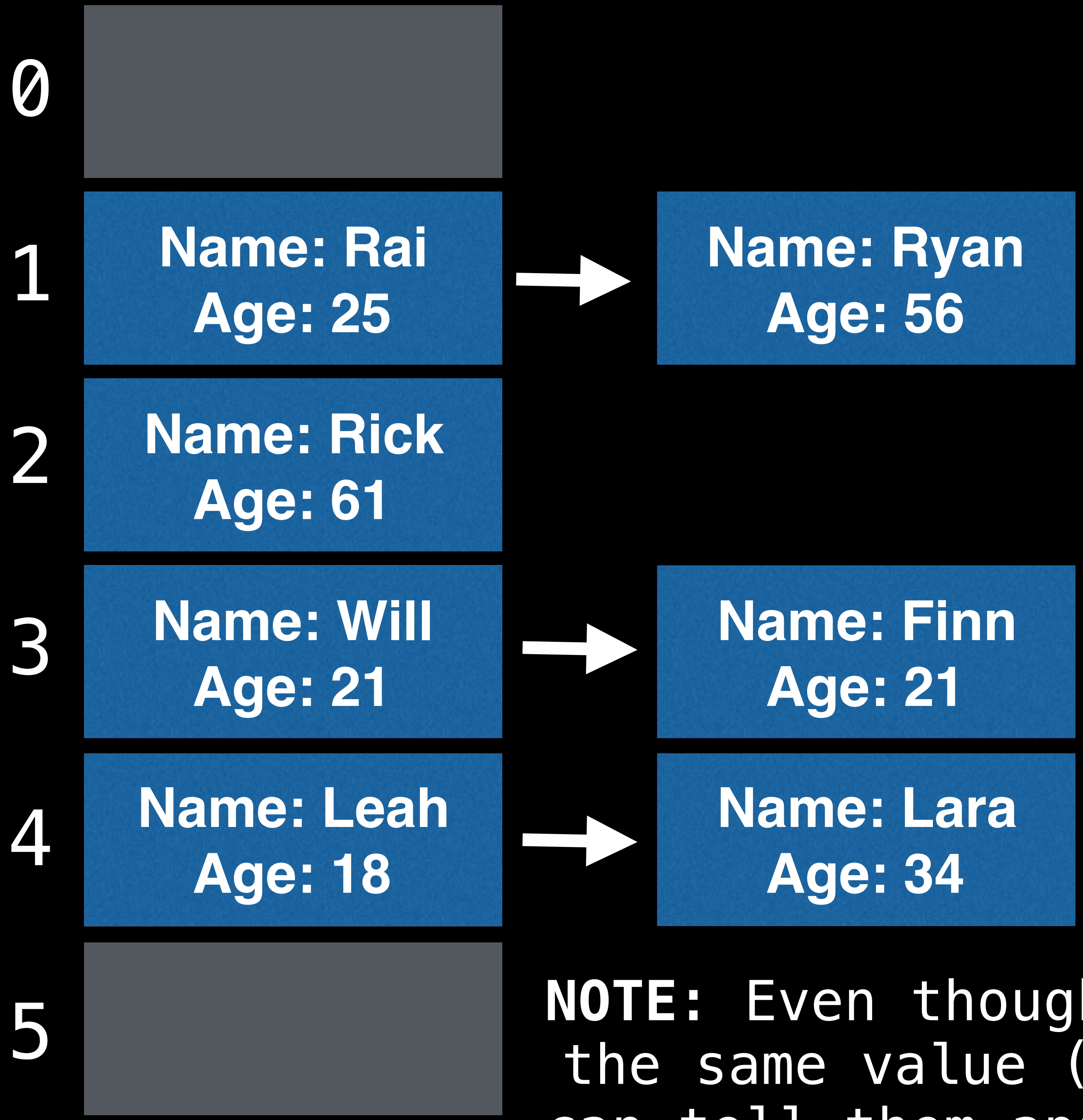
5

Name: Finn

Age: 21

Hash: 3

Linked list Separate Chaining Insertion



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

3 Name: Will Age: 21 → Name: Finn Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34

5

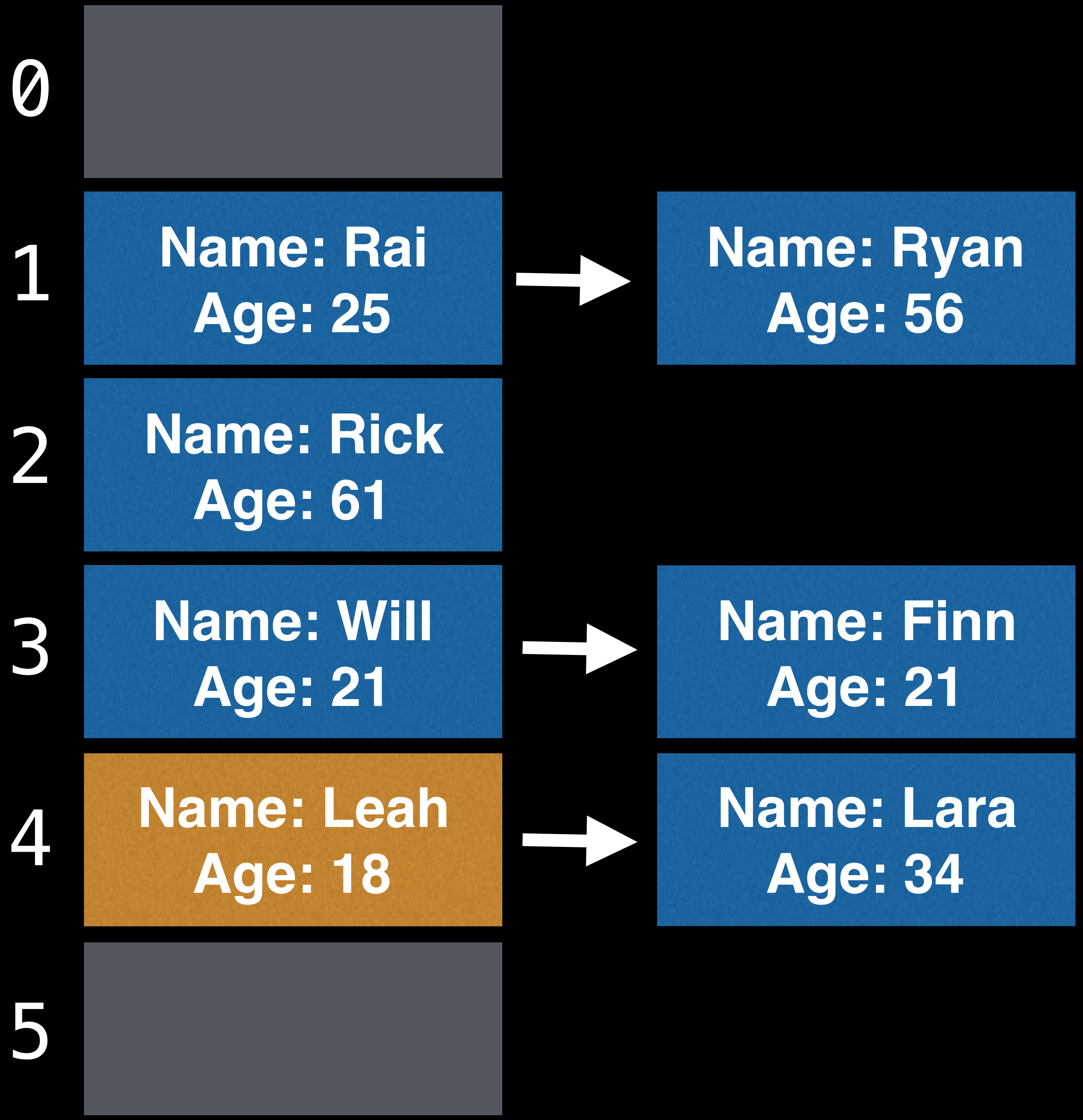
Name: Finn

Age: 21

Hash: 3

NOTE: Even though Finn and Will both hash to the same value (3) and have the same age we can tell them apart because we store the key (name) value in the linked list block.

Linked list Separate Chaining Insertion



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

3 Name: Will Age: 21 → Name: Finn Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34

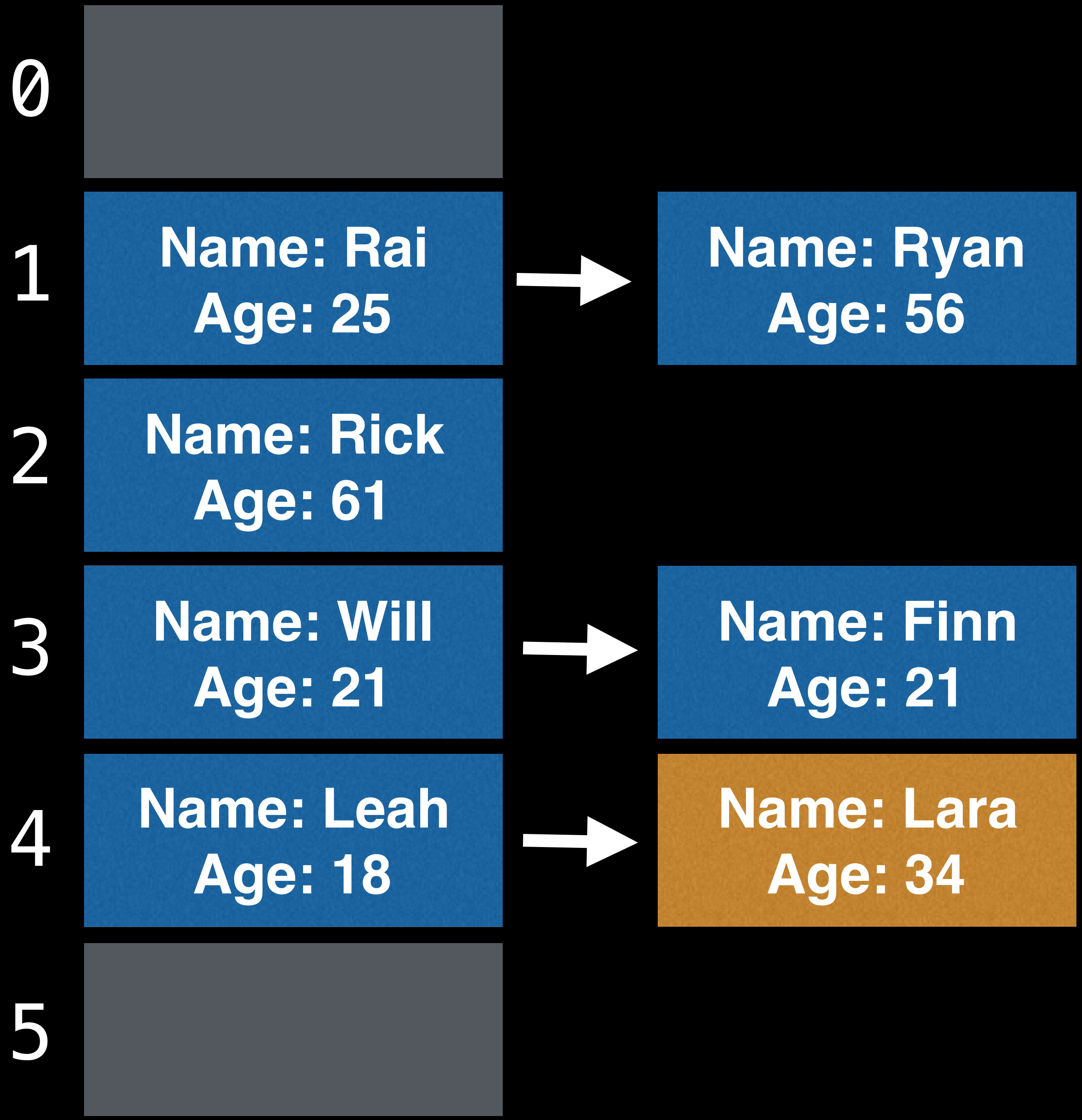
5

Name: Mark

Age: 10

Hash: 4

Linked list Separate Chaining Insertion



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

3 Name: Will Age: 21 → Name: Finn Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34

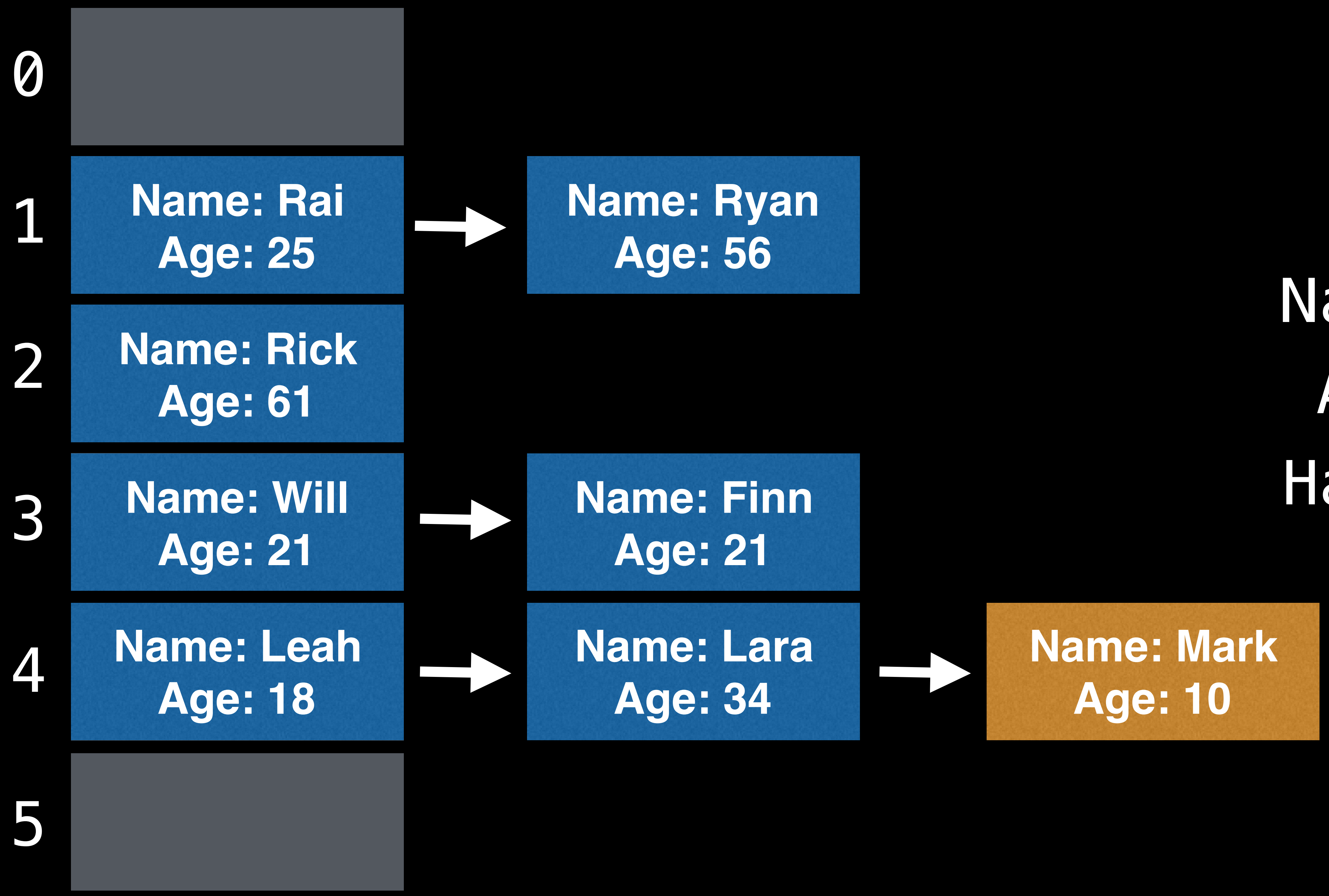
5

Name: Mark

Age: 10

Hash: 4

Linked list Separate Chaining Insertion

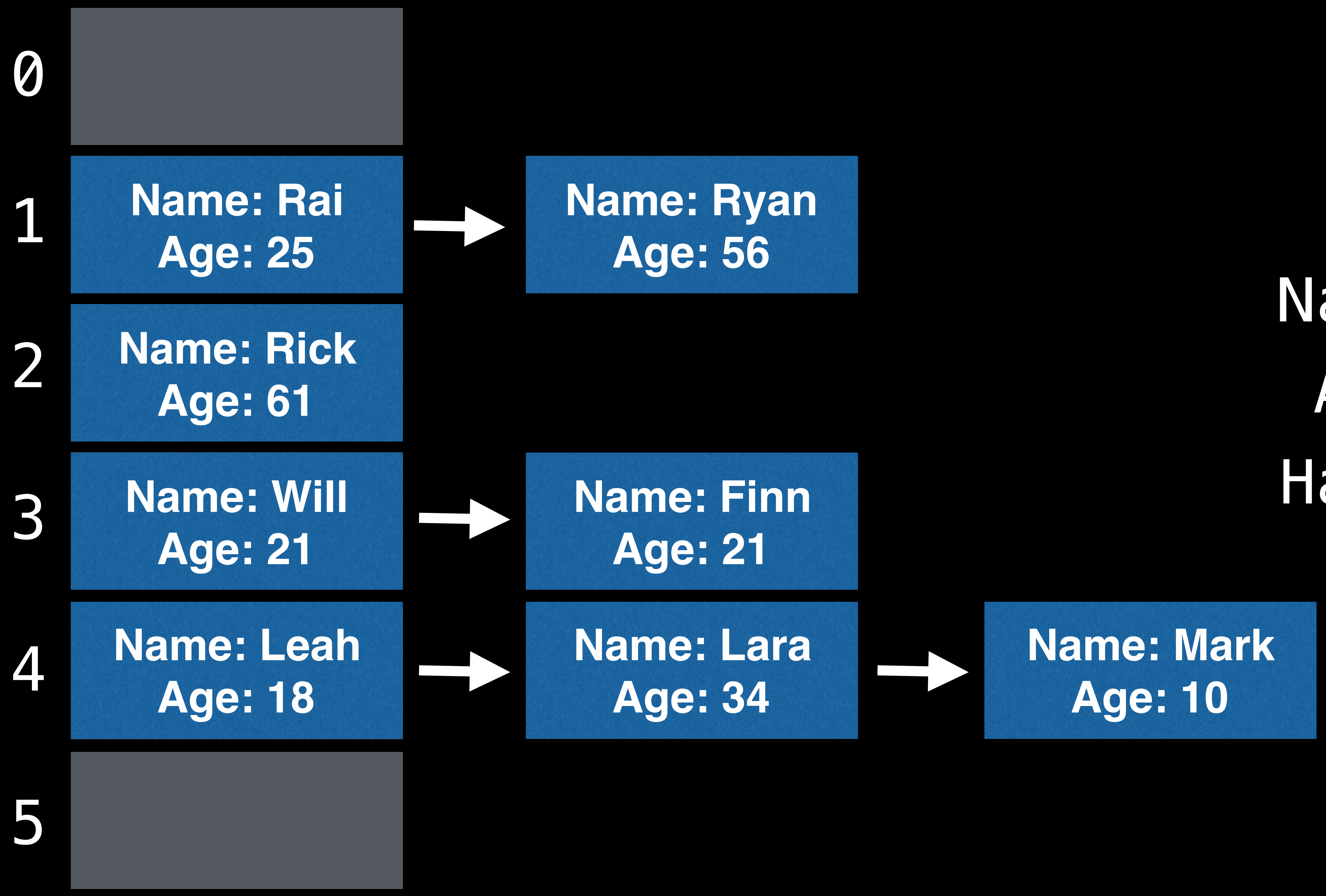


Name: Mark

Age: 10

Hash: 4

Linked list Separate Chaining Insertion



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

3 Name: Will Age: 21 → Name: Finn Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34 → Name: Mark Age: 10

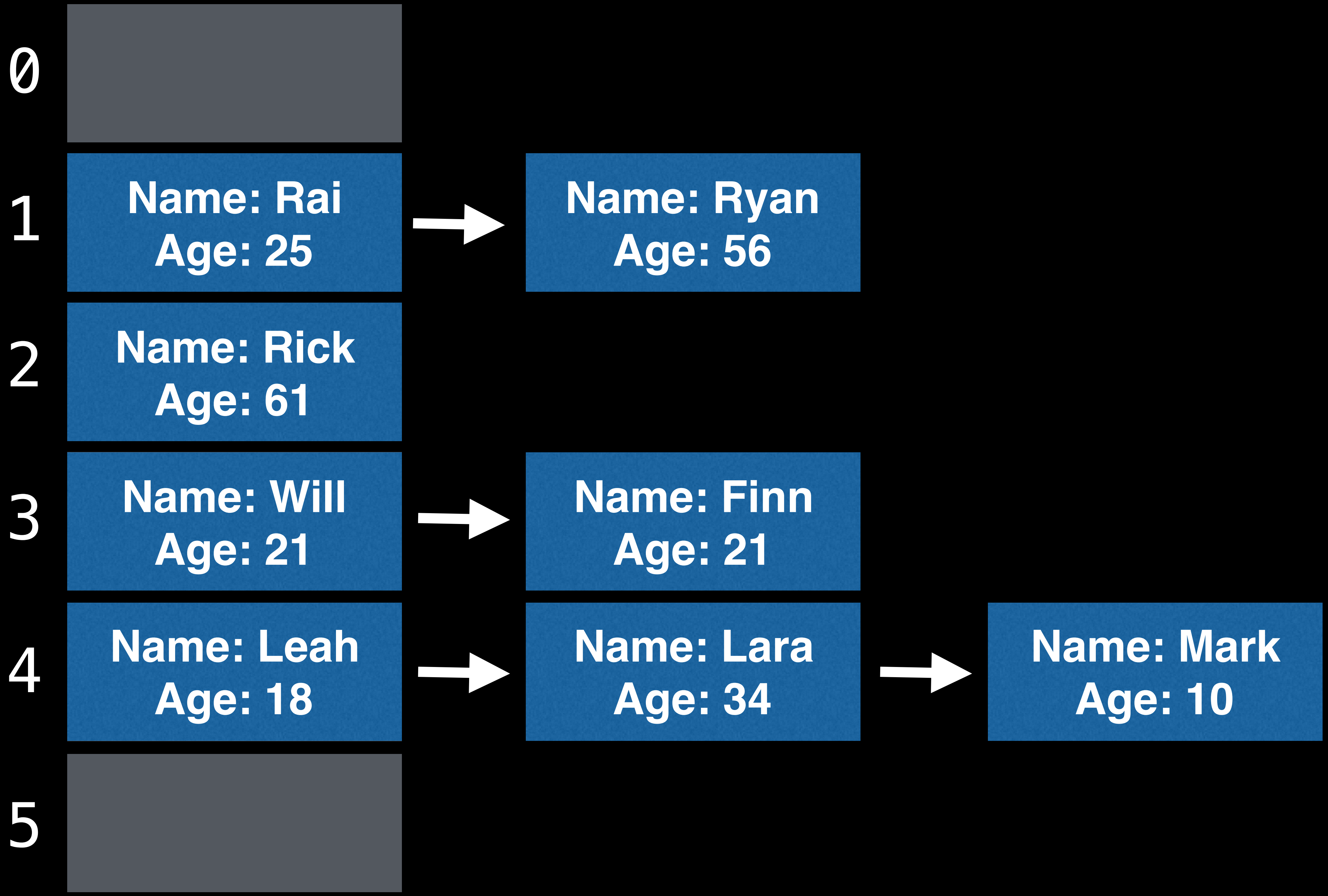
5

Name: Mark

Age: 10

Hash: 4

Linked list Separate Chaining Lookups



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

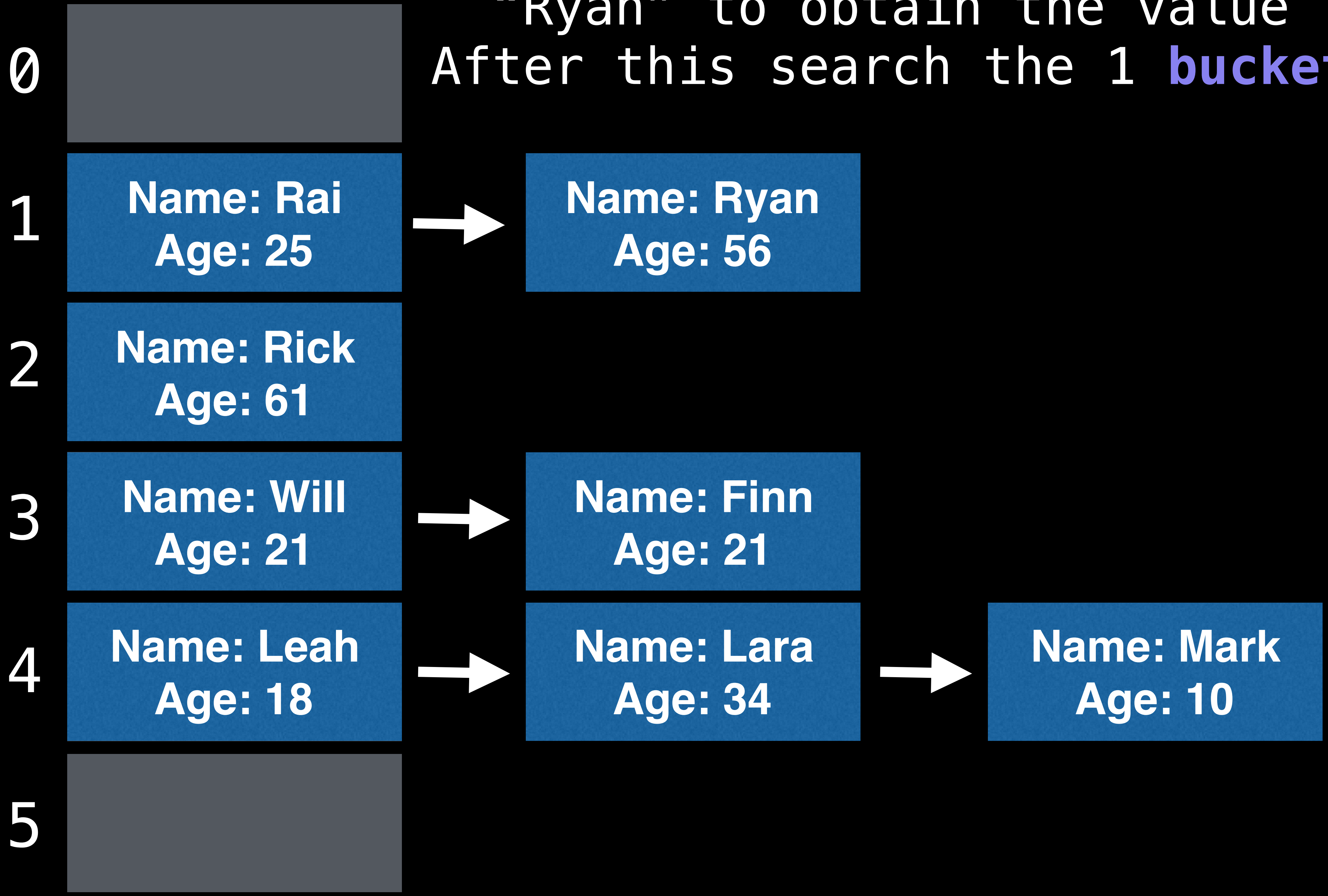
3 Name: Will Age: 21 → Name: Finn Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34 → Name: Mark Age: 10

5

Linked list Separate Chaining Lookups

To find the age of “Ryan” hash the key “Ryan” to obtain the value (index) 1. After this search the 1 **bucket** for “Ryan”



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

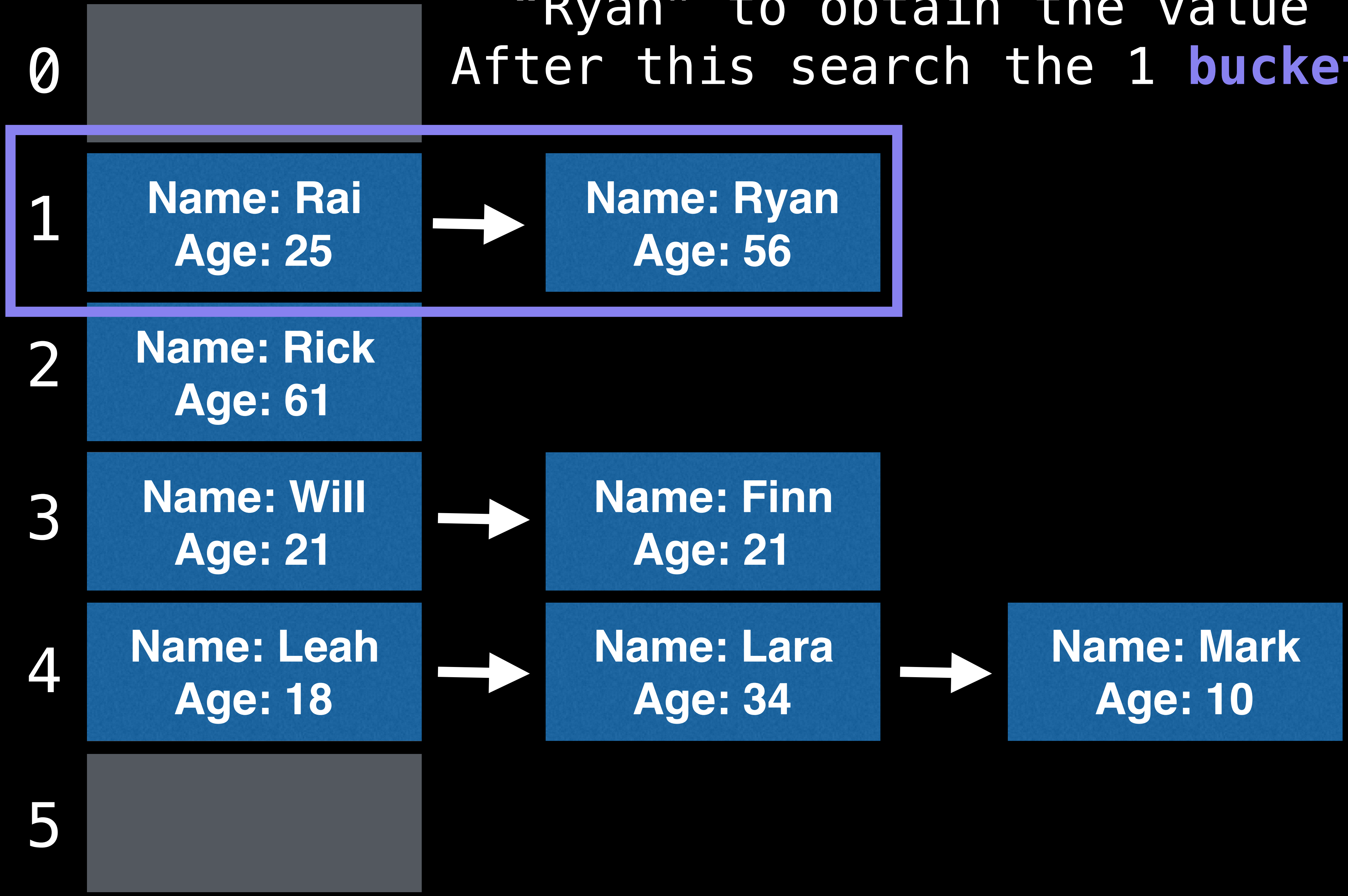
3 Name: Will Age: 21 → Name: Finn Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34 → Name: Mark Age: 10

5

Linked list Separate Chaining Lookups

To find the age of “Ryan” hash the key “Ryan” to obtain the value (index) 1. After this search the 1 **bucket** for “Ryan”



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

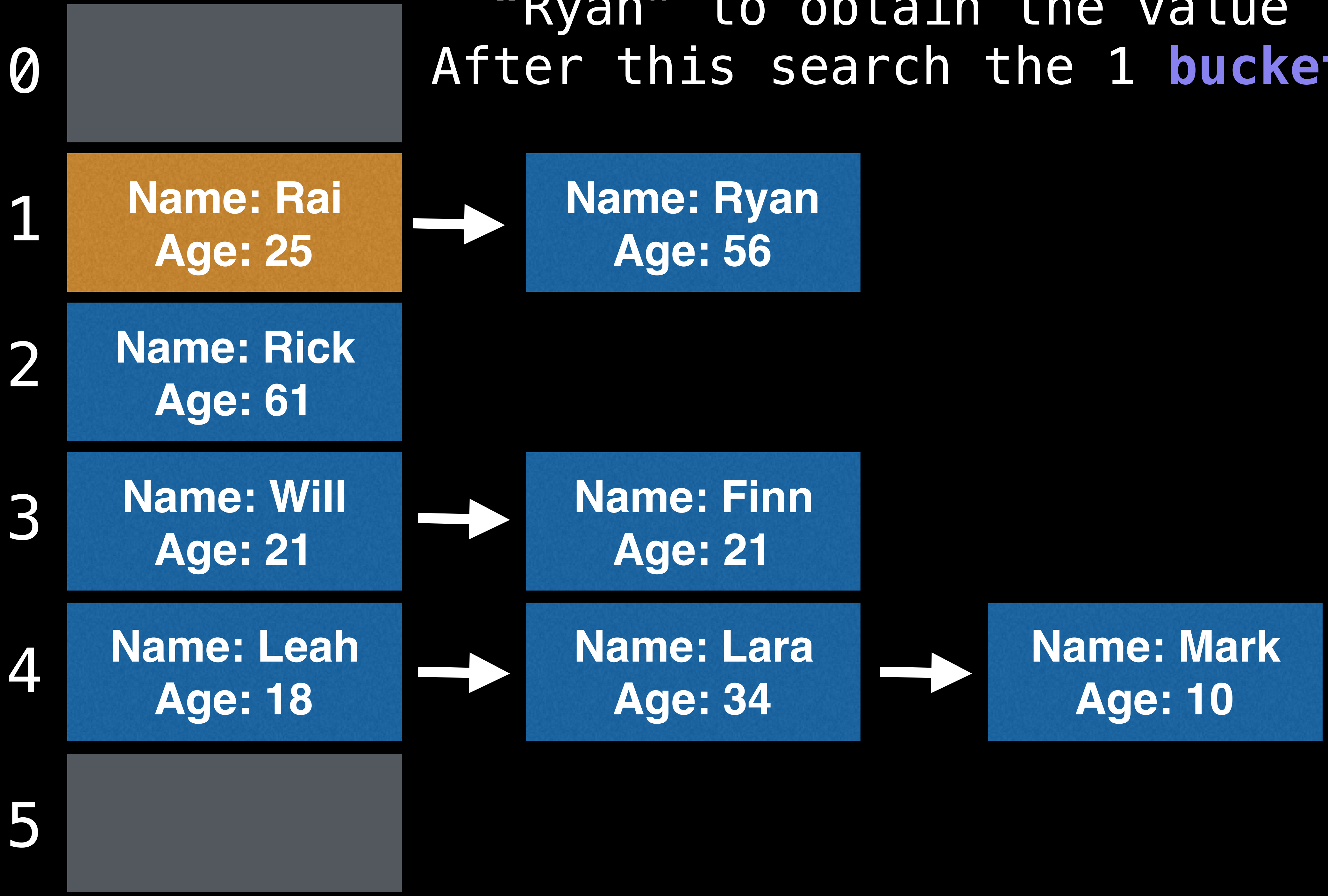
3 Name: Will Age: 21 → Name: Finn Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34 → Name: Mark Age: 10

5

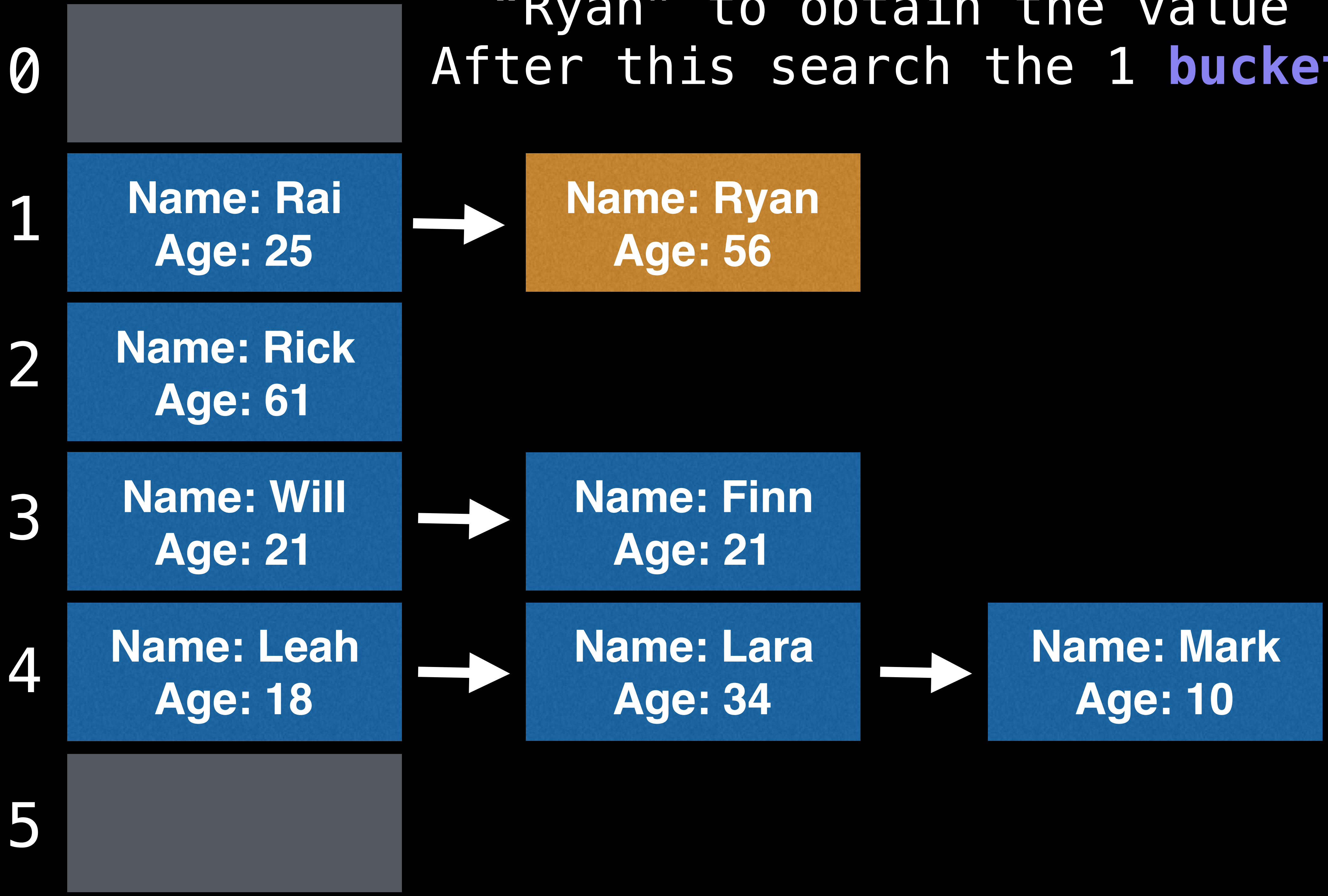
Linked list Separate Chaining Lookups

To find the age of “Ryan” hash the key “Ryan” to obtain the value (index) 1. After this search the 1 **bucket** for “Ryan”



Linked list Separate Chaining Lookups

To find the age of “Ryan” hash the key “Ryan” to obtain the value (index) 1. After this search the 1 **bucket** for “Ryan”



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

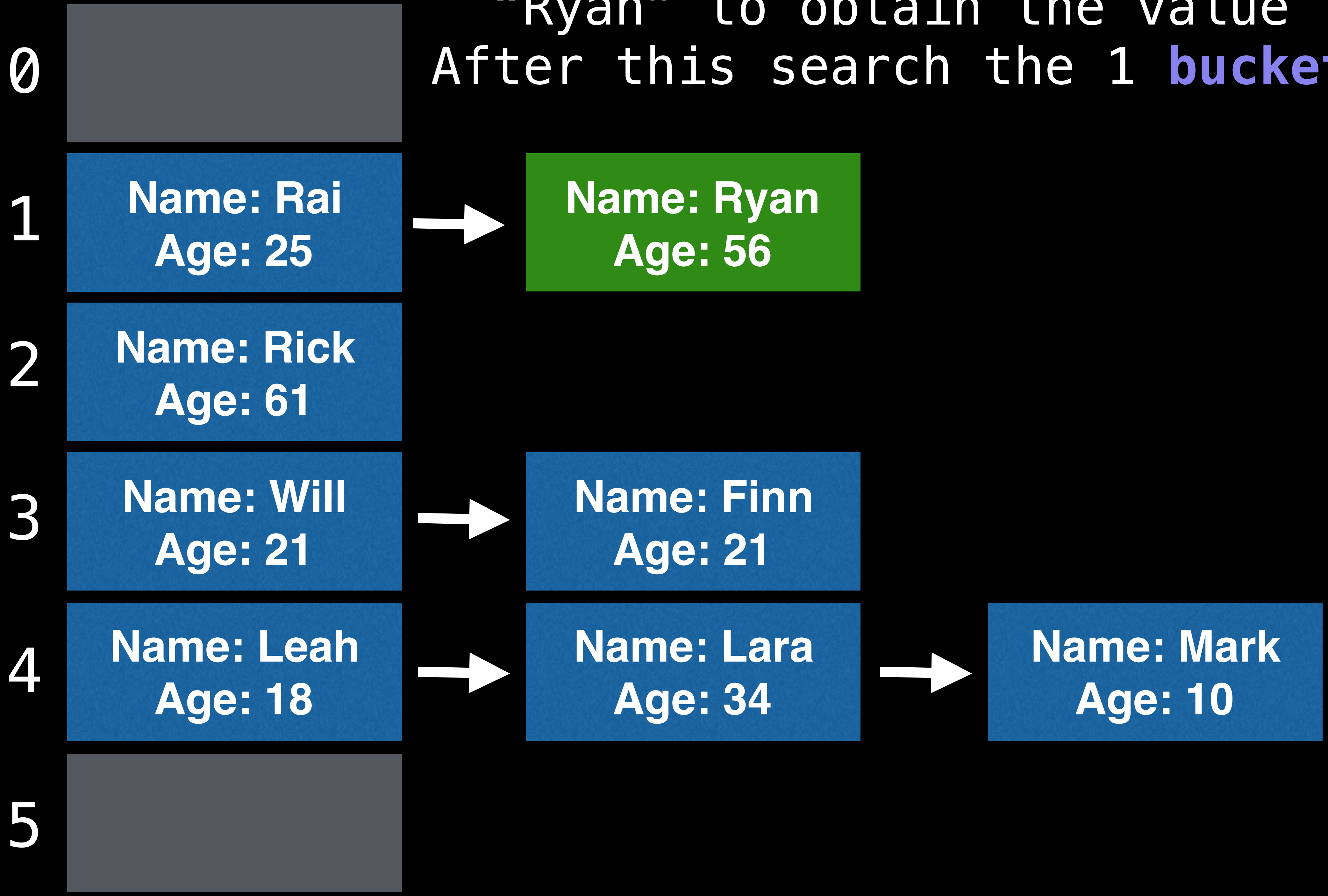
3 Name: Will Age: 21 → Name: Finn Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34 → Name: Mark Age: 10

5

Linked list Separate Chaining Lookups

To find the age of “Ryan” hash the key “Ryan” to obtain the value (index) 1. After this search the 1 **bucket** for “Ryan”



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

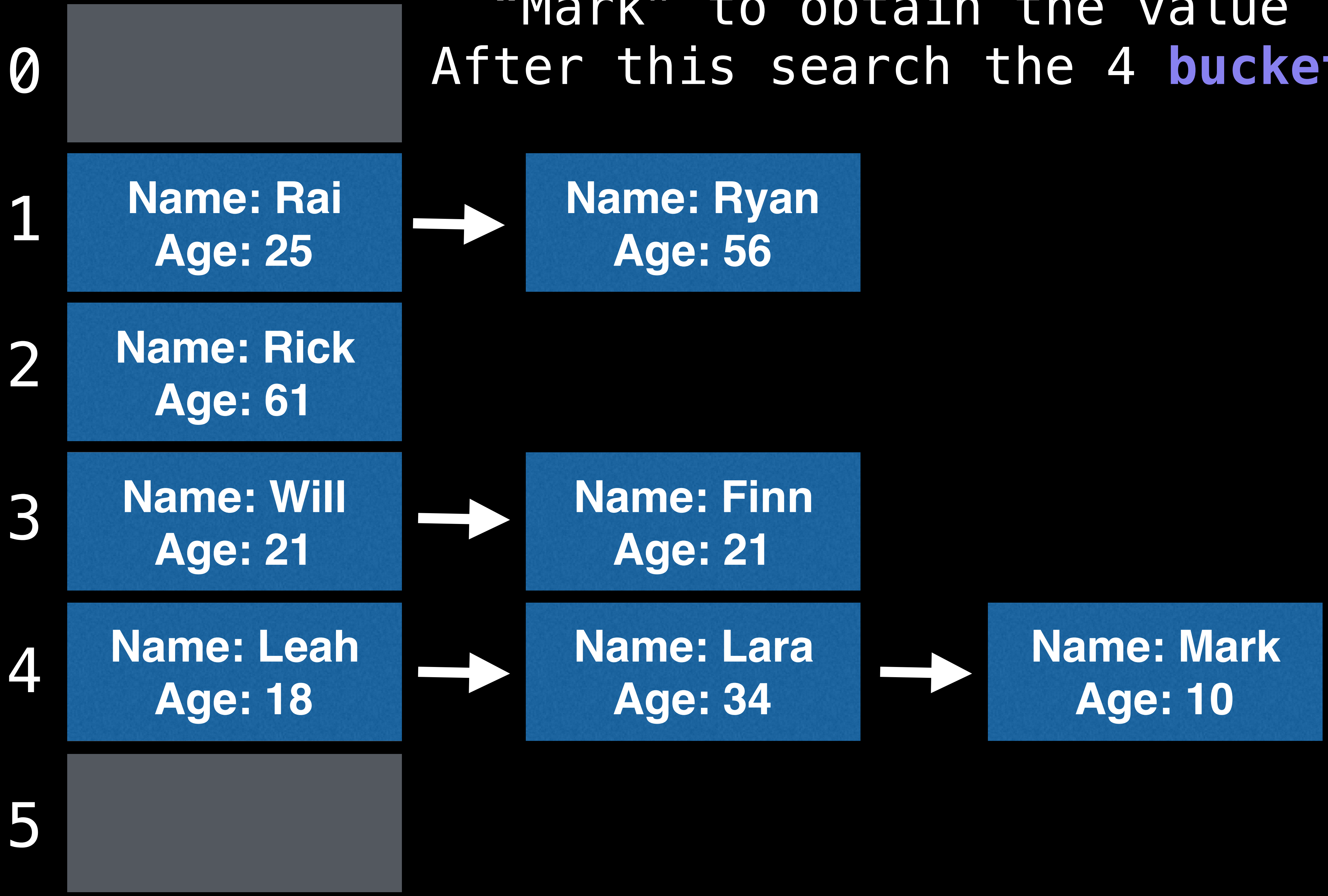
3 Name: Will Age: 21 → Name: Finn Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34 → Name: Mark Age: 10

5

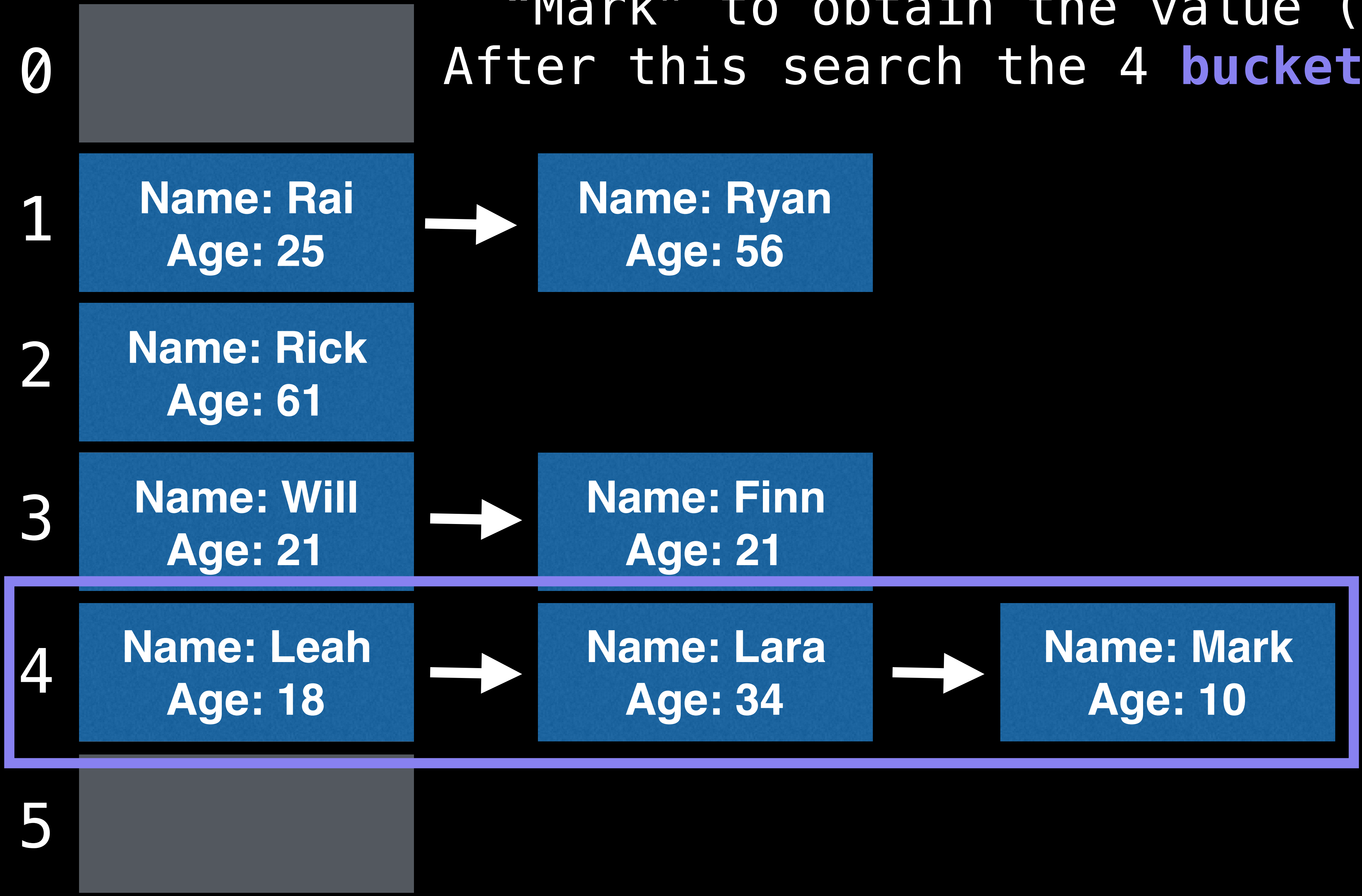
Linked list Separate Chaining Lookups

To find the age of “Mark” hash the key “Mark” to obtain the value (index) 4. After this search the 4 **bucket** for “Mark”



Linked list Separate Chaining Lookups

To find the age of “Mark” hash the key “Mark” to obtain the value (index) 4. After this search the 4 **bucket** for “Mark”



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

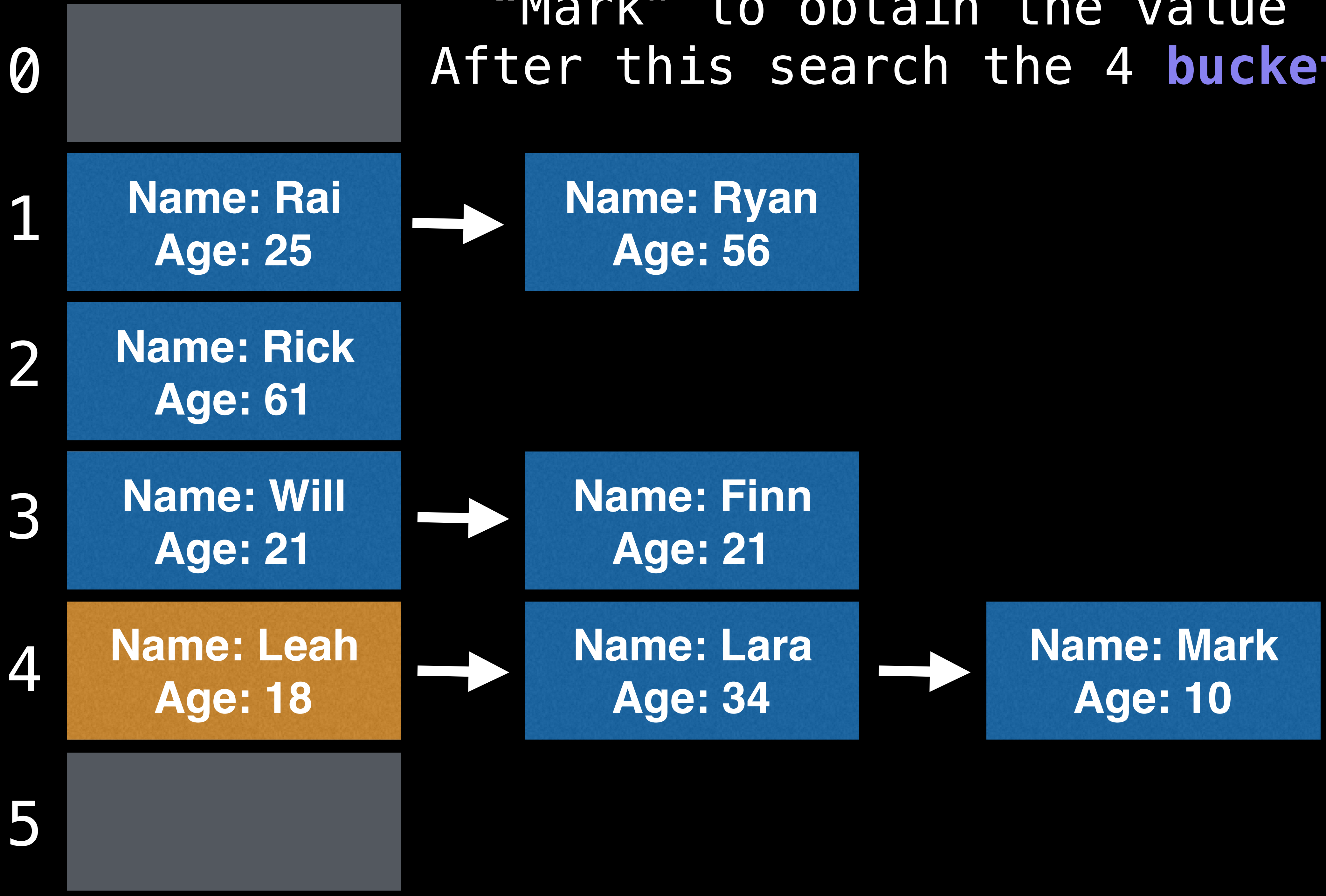
3 Name: Will Age: 21 → Name: Finn Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34 → Name: Mark Age: 10

5

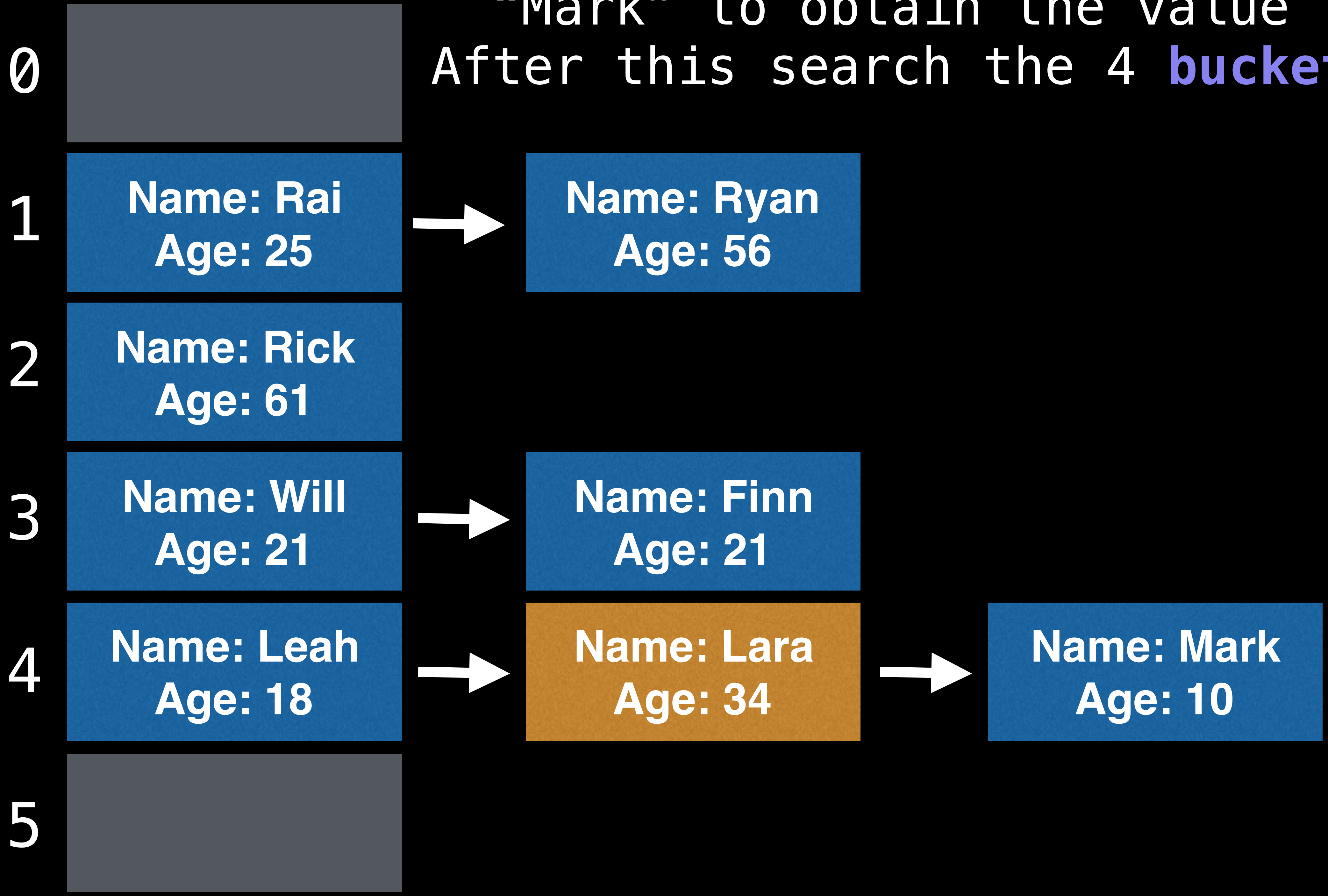
Linked list Separate Chaining Lookups

To find the age of “Mark” hash the key “Mark” to obtain the value (index) 4. After this search the 4 **bucket** for “Mark”



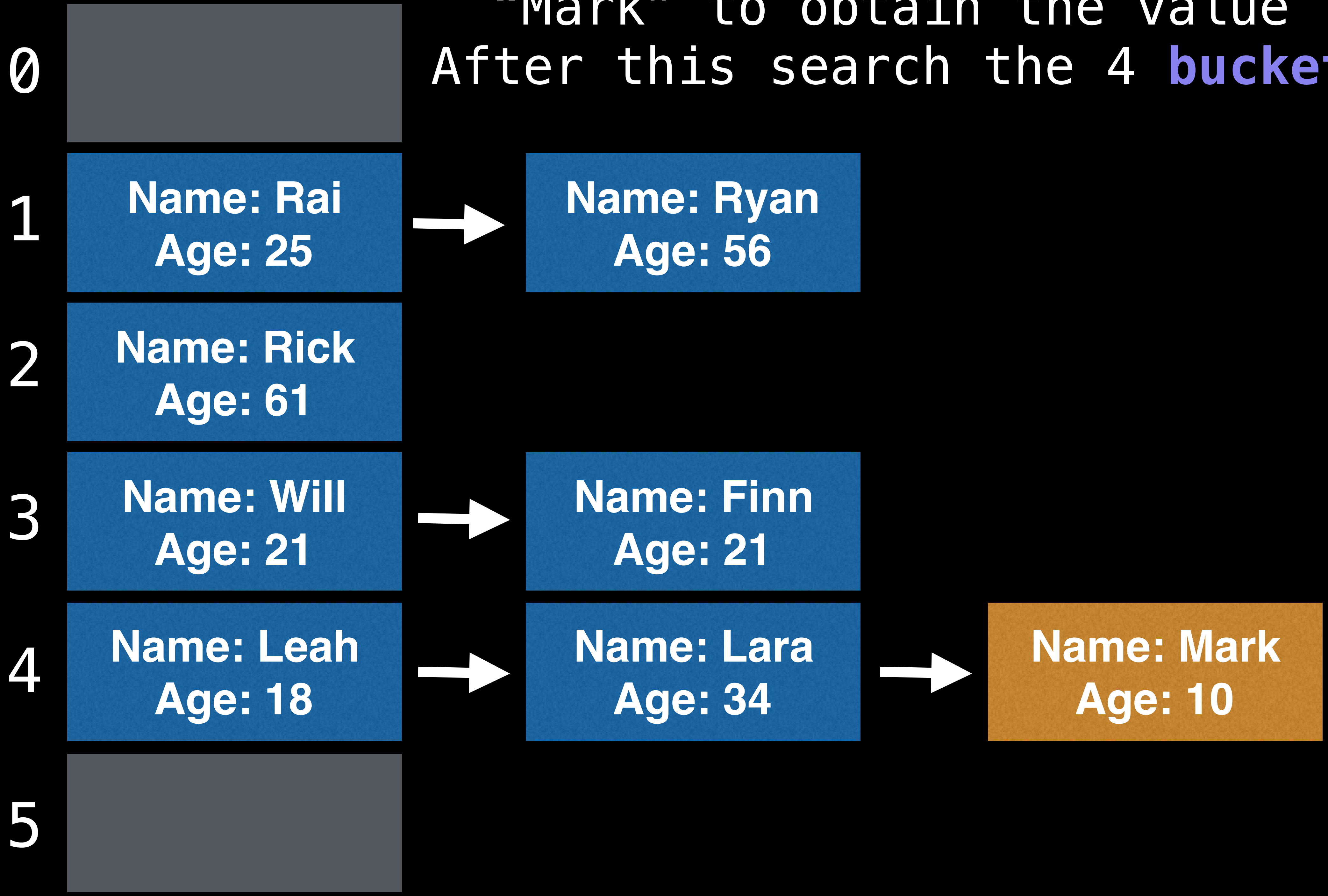
Linked list Separate Chaining Lookups

To find the age of “Mark” hash the key “Mark” to obtain the value (index) 4. After this search the 4 **bucket** for “Mark”



Linked list Separate Chaining Lookups

To find the age of “Mark” hash the key “Mark” to obtain the value (index) 4. After this search the 4 **bucket** for “Mark”



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

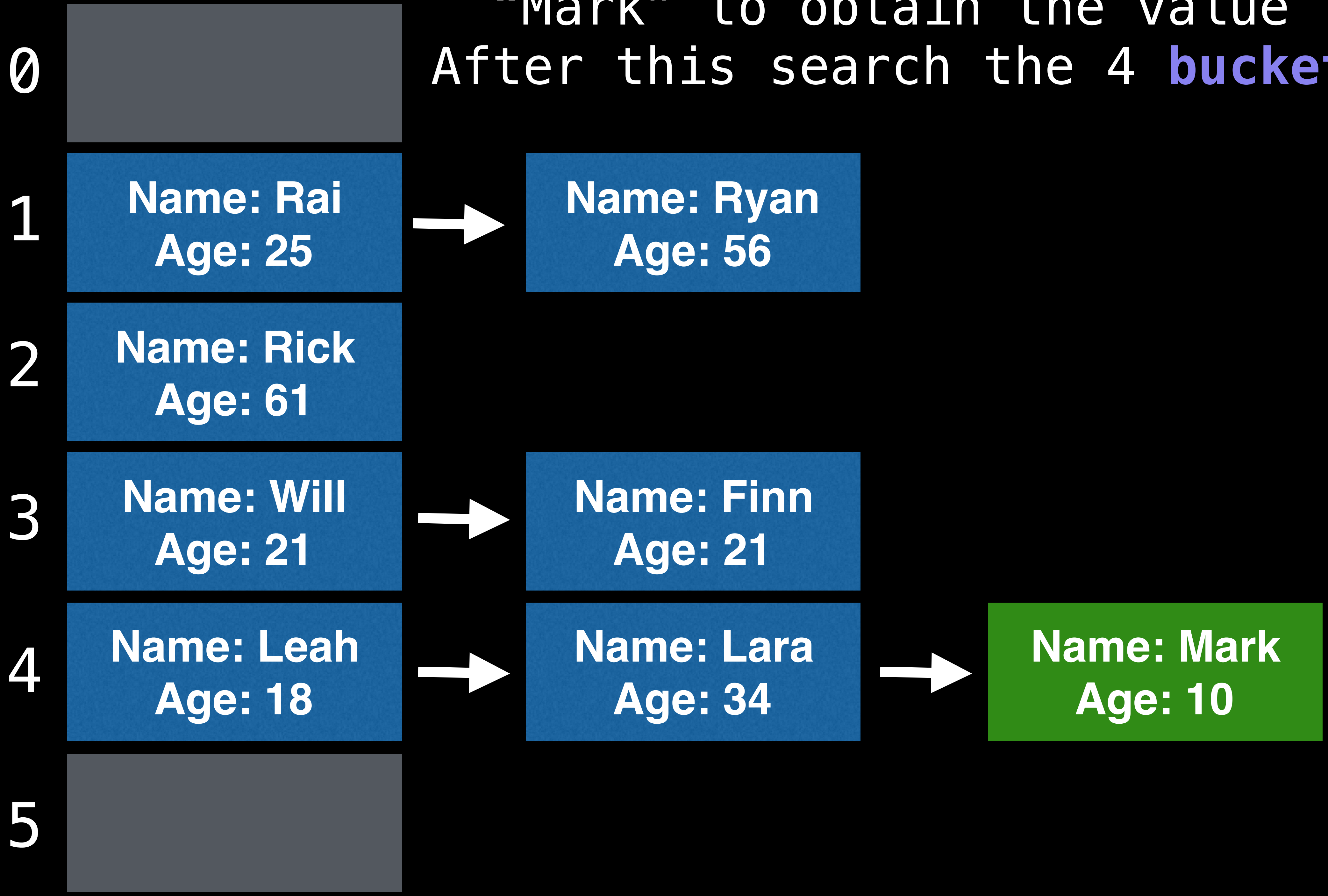
3 Name: Will Age: 21 → Name: Finn Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34 → Name: Mark Age: 10

5

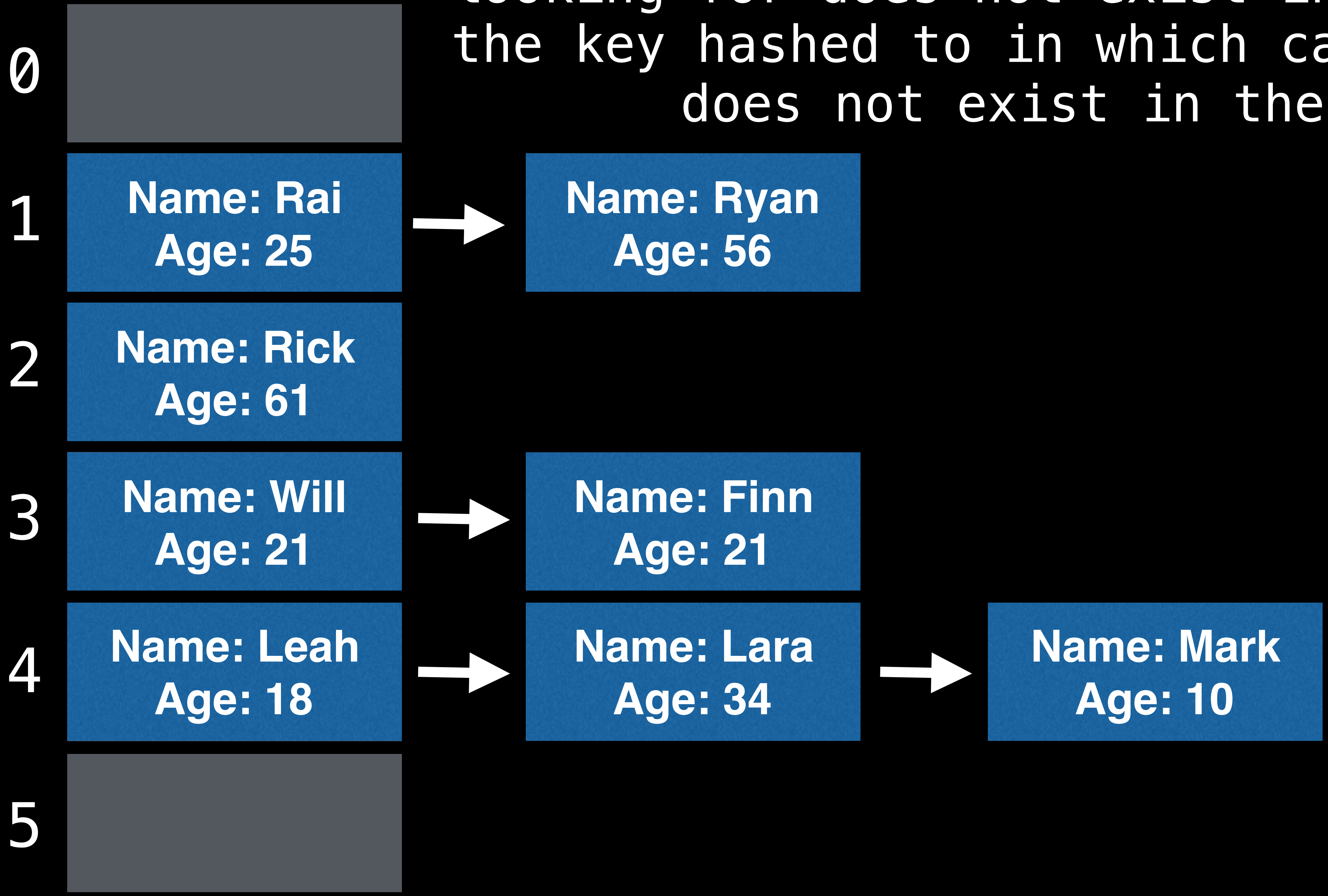
Linked list Separate Chaining Lookups

To find the age of “Mark” hash the key “Mark” to obtain the value (index) 4. After this search the 4 **bucket** for “Mark”



Linked list Separate Chaining Lookups

It may happen that the value you are looking for does not exist in the bucket the key hashed to in which case the item does not exist in the HT.



0

1 Name: Rai Age: 25 → Name: Ryan Age: 56

2 Name: Rick Age: 61

3 Name: Will Age: 21 → Name: Finn Age: 21

4 Name: Leah Age: 18 → Name: Lara Age: 34 → Name: Mark Age: 10

5

Hash table FAQs

Q: How do I maintain **O(1)** insertion and lookup time complexity once my HT gets really full and I have long linked list chains?

A: Once the HT contains a lot of elements you should create a new HT with a larger capacity and rehash all the items inside the old HT and disperse them throughout the new HT at different locations.

Hash table FAQs

Q: How do I **remove** key-value pairs from my HT?

A: Apply the same procedure as doing a lookup for a key, but this time instead of returning the value associated with the key remove the node in the linked list data structure.

Hash table FAQs

Q: Can I use another data structure to model the bucket behaviour required for the separate chaining method?

A: Of course! Common data structures used instead of a linked list include: **arrays**, **binary trees**, **self balancing trees**, etc… You can even go with a hybrid approach like Java’s HashMap. However, note that some of these are much more memory intensive and complex to implement than a simple linked list which is why they may be less popular.

Next Video:
Hash tables with open addressing!

Hash table separate chaining implementation and source code and tests can all be found at:

github.com/williamfiset/data-structures

Hash table (HT) open addressing

William Fiset

Open addressing basics

The goal of the **Hash Table (HT)** is to construct a **mapping** from keys to values.

Keys must be **hashable** and we need a **hash function** that converts keys to whole numbers.

We use the hash function defined on our key set to **index into** an array (the hash table).

Hash functions are not perfect, therefore sometimes two keys k_1, k_2 ($k_1 \neq k_2$) hash to the same value. When this happens we have a **hash collision** (i.e $H(k_1) = H(k_2)$)

Open addressing is a way to solve this issue.

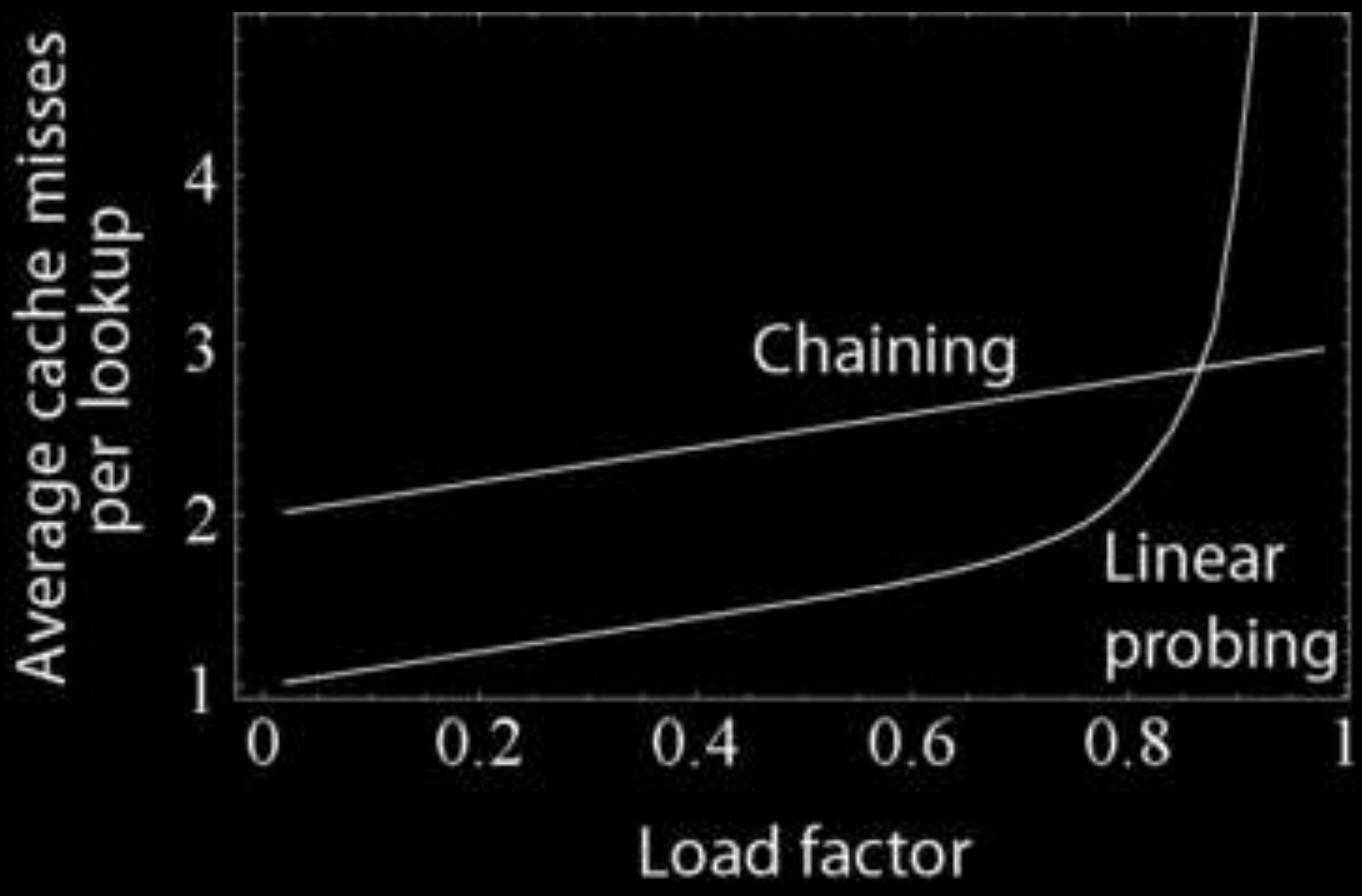
Open addressing basics

When using open addressing as a collision resolution technique the **key-value pairs are stored in the table itself** as opposed to a data structure like in separate chaining.

This means we need to care a great deal about the size of our hash table and how many elements are currently in the table.

$$\textbf{Load factor} = \frac{\textbf{items in table}}{\textbf{size of table}}$$

Open addressing basics



Source: Wikipedia

The **O(1)** constant time behaviour attributed to hash tables assumes the load factor (α) is kept below a certain fixed value. This means once α > **threshold** we need to grow the table size (ideally exponentially, e.g. double).

Open addressing main idea

When we want to insert a key-value pair (k,v) into the hash table we hash the key and obtain an original position for where this key-value pair belongs, i.e **H**(k).

If the position our key hashed to is occupied, try another position in the hash table by offsetting the current position subject to a **probing sequence P**(x). Keep doing this until an unoccupied slot is found.

Open addressing main idea

There are an infinite amount of probing sequences you can come up with, here are a few:

Linear probing:

P(x) = ax + b where a, b are constants

Quadratic probing:

P(x)= ax^2 + bx + c, where a,b,c are constants

Double hashing:

P(k,x) = x*H_2(k), where H_2(k) is a secondary hash function

Pseudo random number generator:

P(k,x) = x***RNG**(**H**(k),x), where **RNG** is a random number generator function seeded with **H**(k).

Open addressing main idea

General insertion method for open addressing on a table of size N goes as follows:

```
x := 1
keyHash := H(k) mod N
index := keyHash

while table[index] != null:
    index = (keyHash + P(k,x)) mod N
    x = x + 1

insert (k,v) at table[index]
```

Where **H**(k) is the hash for the key k and **P**(k,x) is the probing function

Chaos with cycles

Most randomly selected probing sequences modulo N will produce a cycle shorter than the table size.

This becomes problematic when you are trying to insert a key-value pair and all the buckets on the cycle are occupied because you will get stuck in an **infinite loop**!

Chaos with cycles

0	1	2	3	4	5	6	7	8	9	10	11
k_1,v_1	∅	∅	k_2,v_2	k_3,v_3	k_4,v_4	∅	∅	k_5,v_5	∅	k_6,v_6	∅

Suppose we have a hash table of size 12 which is already partially full. The occupied cells are filled with a key-value pairs (k_i,v_i) and empty cells with a null token: ∅

Chaos with cycles

0	1	2	3	4	5	6	7	8	9	10	11
k_1,v_1	∅	∅	k_2,v_2	k_3,v_3	k_4,v_4	∅	∅	k_5,v_5	∅	k_6,v_6	∅

Assume the probing sequence used is **P**(x) = 4x

Now suppose we want to insert (k,v)
into the table and **H**(k) = 8

Chaos with cycles

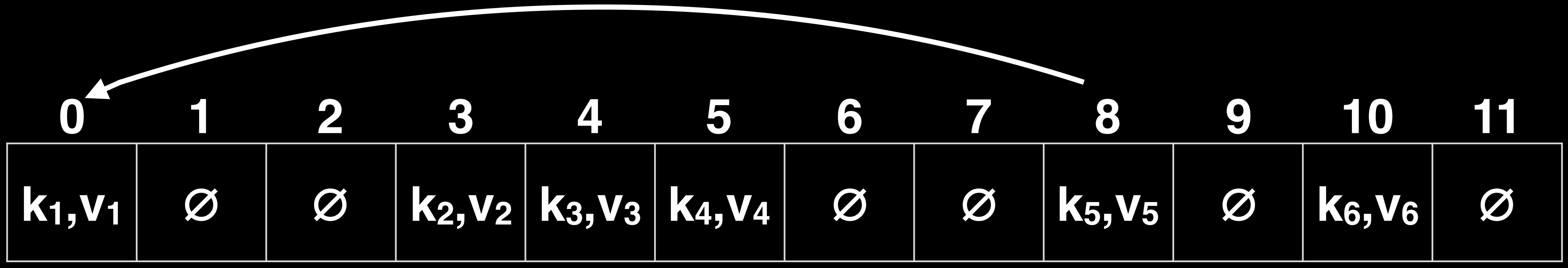
0	1	2	3	4	5	6	7	8	9	10	11
k_1,v_1	∅	∅	k_2,v_2	k_3,v_3	k_4,v_4	∅	∅	k_5,v_5	∅	k_6,v_6	∅
								↑			

Assume the probing sequence used is **P**(x) = 4x

Now suppose we want to insert (k,v)
into the table and **H**(k) = 8

index = **H**(k) = 8 + 0 mod 12 = 8

Chaos with cycles



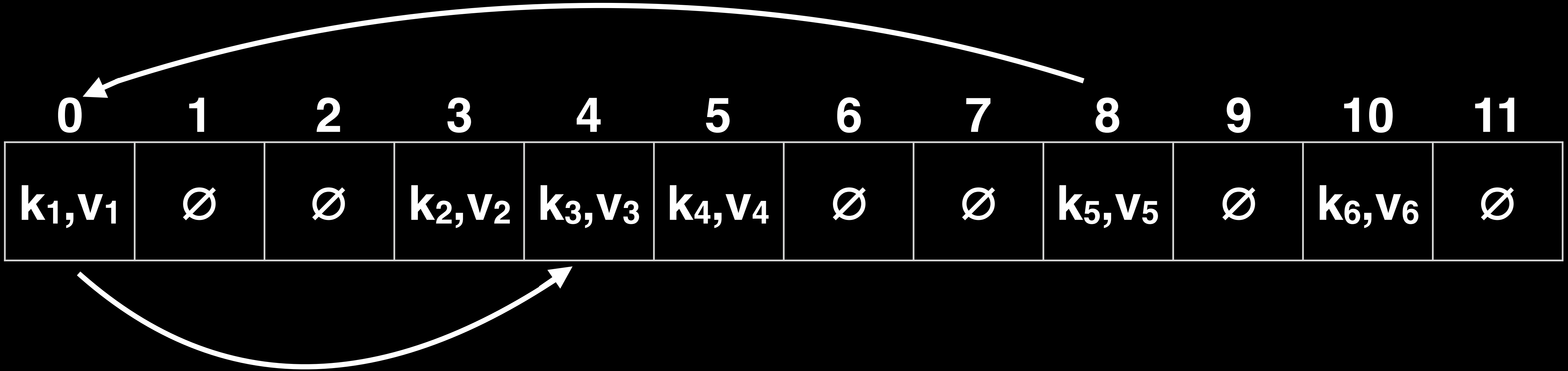
0	1	2	3	4	5	6	7	8	9	10	11
k_1,v_1	∅	∅	k_2,v_2	k_3,v_3	k_4,v_4	∅	∅	k_5,v_5	∅	k_6,v_6	∅

Assume the probing sequence used is P(x) = 4x

Now suppose we want to insert (k,v)
into the table and H(k) = 8

index = H(k) = 8 + 0 mod 12 = 8
index = H(k) + P(1) = 8 + 4 mod 12 = 0

Chaos with cycles



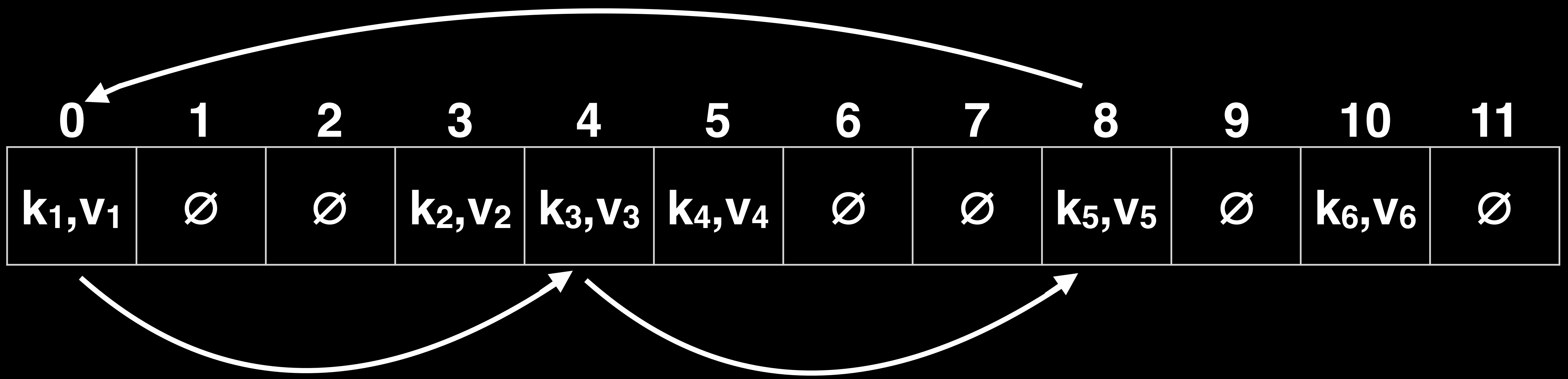
0	1	2	3	4	5	6	7	8	9	10	11
k_1,v_1	∅	∅	k_2,v_2	k_3,v_3	k_4,v_4	∅	∅	k_5,v_5	∅	k_6,v_6	∅

Assume the probing sequence used is **P**(x) = 4x

Now suppose we want to insert (k,v)
into the table and **H**(k) = 8

index = **H**(k) = 8 + 0 mod 12 = 8
index = **H**(k) + **P**(1) = 8 + 4 mod 12 = 0
index = **H**(k) + **P**(2) = 8 + 8 mod 12 = 4

Chaos with cycles



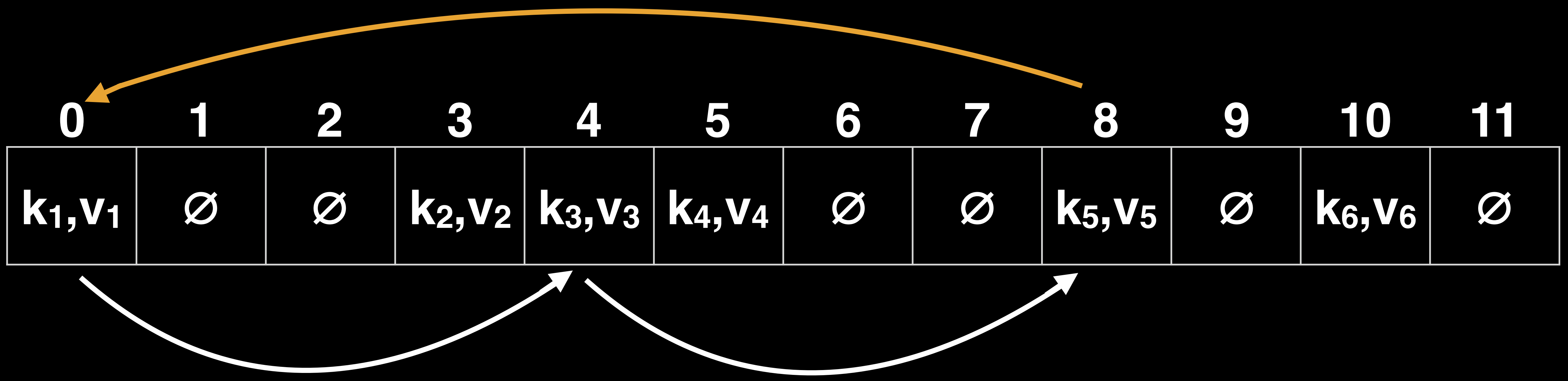
0	1	2	3	4	5	6	7	8	9	10	11
k_1,v_1	∅	∅	k_2,v_2	k_3,v_3	k_4,v_4	∅	∅	k_5,v_5	∅	k_6,v_6	∅

Assume the probing sequence used is P(x) = 4x

Now suppose we want to insert (k,v)
into the table and H(k) = 8

index = H(k)   = 8 + 0 mod 12 = 8
index = H(k) + P(1) = 8 + 4 mod 12 = 0
index = H(k) + P(2) = 8 + 8 mod 12 = 4
index = H(k) + P(3) = 8 + 12 mod 12 = 8

Chaos with cycles



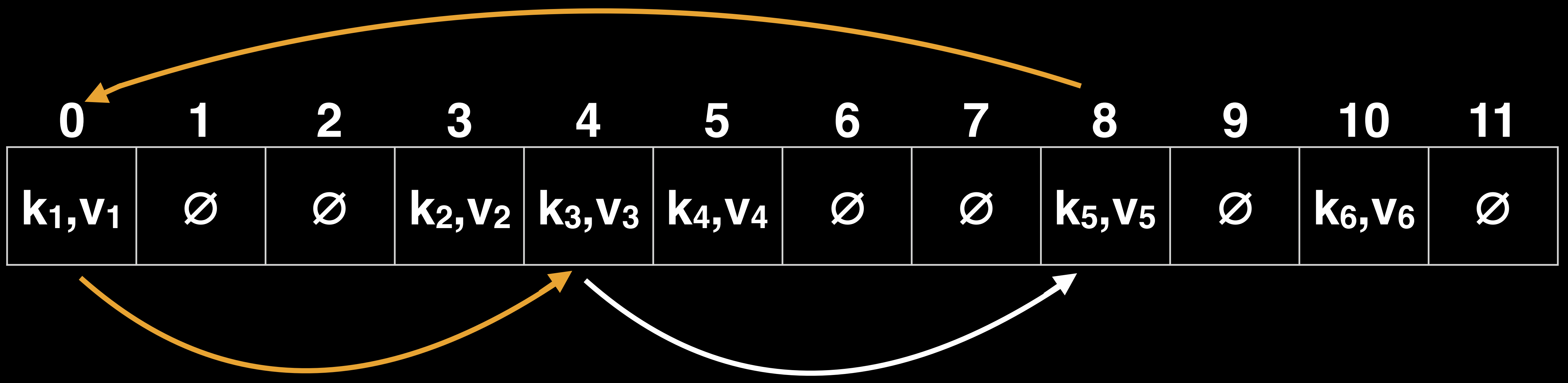
0	1	2	3	4	5	6	7	8	9	10	11
k_1,v_1	∅	∅	k_2,v_2	k_3,v_3	k_4,v_4	∅	∅	k_5,v_5	∅	k_6,v_6	∅

Assume the probing sequence used is P(x) = 4x

Now suppose we want to insert (k,v)
into the table and H(k) = 8

index = H(k) = 8 + 0 mod 12 = 8
index = H(k) + P(1) = 8 + 4 mod 12 = 0
index = H(k) + P(2) = 8 + 8 mod 12 = 4
index = H(k) + P(3) = 8 + 12 mod 12 = 8
index = H(k) + P(4) = 8 + 16 mod 12 = 0

Chaos with cycles



0	1	2	3	4	5	6	7	8	9	10	11
k_1,v_1	∅	∅	k_2,v_2	k_3,v_3	k_4,v_4	∅	∅	k_5,v_5	∅	k_6,v_6	∅

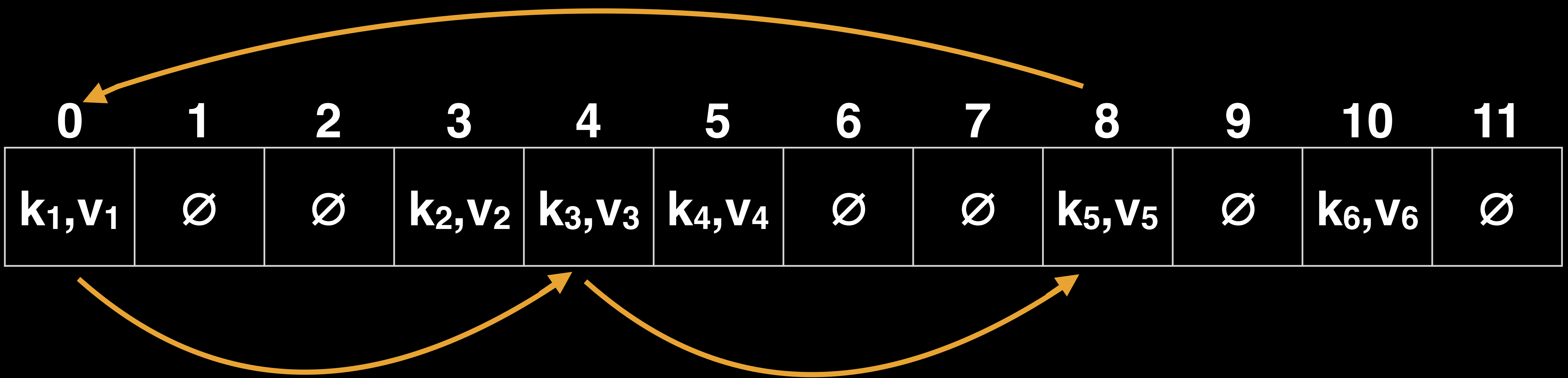
Assume the probing sequence used is P(x) = 4x

Now suppose we want to insert (k,v)
into the table and H(k) = 8

index = H(k) = 8 + 0 mod 12 = 8
index = H(k) + P(1) = 8 + 4 mod 12 = 0
index = H(k) + P(2) = 8 + 8 mod 12 = 4
index = H(k) + P(3) = 8 + 12 mod 12 = 8
index = H(k) + P(4) = 8 + 16 mod 12 = 0
index = H(k) + P(5) = 8 + 20 mod 12 = 4

…

Chaos with cycles



0	1	2	3	4	5	6	7	8	9	10	11
k_1,v_1	∅	∅	k_2,v_2	k_3,v_3	k_4,v_4	∅	∅	k_5,v_5	∅	k_6,v_6	∅

Assume the probing sequence used is **P**(x) = 4x

Now suppose we want to insert (k,v)
into the table and **H**(k) = 8

```
index = H(k)        = 8 + 0  mod 12 = 8
index = H(k) + P(1) = 8 + 4  mod 12 = 0
index = H(k) + P(2) = 8 + 8  mod 12 = 4
index = H(k) + P(3) = 8 + 12 mod 12 = 8
index = H(k) + P(4) = 8 + 16 mod 12 = 0
index = H(k) + P(5) = 8 + 20 mod 12 = 4
```

…

Chaos with cycles

Q: So that’s concerning… how do we handle probing functions which produce cycles shorter than the table size?

A: In general the consensus is that we don’t handle this issue. Instead we avoid it altogether by restricting our domain of probing functions to those which produce a cycle of exactly length N*.

* There are a few exceptions with special properties that can produce shorter cycles.

Chaos with cycles

Techniques such as **linear probing**, **quadratic probing** and **double hashing** are all subject to the issue of causing cycles which is why the **probing functions used with these methods are very specific**. This is a large topic that will be the focus of the next few videos.

Notice that open addressing is very sensitive to the hashing function and probing function used. This is not something you have to worry about (as much) if you are using **separate chaining** as a collision resolution method.

Next Video:
Open addressing linear probing

Multiple hash table implementations and source code and tests can all be found at:

github.com/williamfiset/data-structures

Hash table Linear Probing

An in depth look at linear probing

William Fiset

Open addressing main idea

General insertion method for open addressing on a table of size N goes as follows:

```
x := 1
keyHash := H(k) mod N
index := keyHash

while table[index] != null:
    index = (keyHash + P(k,x)) mod N
    x = x + 1

insert (k,v) at table[index]
```

Where H(k) is the hash for the key k and P(k,x) is the probing function

What is Linear Probing (LP)?

LP is a **probing method** which probes according to a linear formula, specifically:

$P(x) = ax + b$ where $a(\neq 0)$, b are constants

(Note: The constant b is obsolete, do you know why?)

However, as we previously saw not all linear functions are viable because they are unable to produce a cycle of order **N**. We will need some way to handle this.

Chaos with cycles

If our linear function is: **P**(x) = 3x, **H**(k) = 4, and table size is nine (N = 9) we end up with the following cycle occurring:

```
H(k)+P(0) mod N = 4
H(k)+P(1) mod N = 7
H(k)+P(2) mod N = 1
H(k)+P(3) mod N = 4
H(k)+P(4) mod N = 7
H(k)+P(5) mod N = 1
H(k)+P(6) mod N = 4
H(k)+P(7) mod N = 7
H(k)+P(8) mod N = 1
          …
```

Chaos with cycles

If our linear function is: P(x) = 3x, H(k) = 4, and table size is nine (N = 9) we end up with the following cycle occurring:

```
H(k)+P(0) mod N = 4
H(k)+P(1) mod N = 7
H(k)+P(2) mod N = 1
H(k)+P(3) mod N = 4
H(k)+P(4) mod N = 7
H(k)+P(5) mod N = 1
H(k)+P(6) mod N = 4
H(k)+P(7) mod N = 7
H(k)+P(8) mod N = 1
          …
```

The cycle {4,7,1} makes it impossible to reach buckets {0,2,3,5,6,8}!

This would cause an **infinite loop** in our hash table if all the buckets 4, 7, and 1 were already occupied!

Chaos with cycles

Q: Which value(s) of the constant a in **P**(x) = ax produce a full cycle modulo N?

Chaos with cycles

Q: Which value(s) of the constant a in **P**(x) = ax produce a full cycle modulo N?

A: This happens when a and N are **relatively prime**. Two numbers are relatively prime if their **Greatest Common Denominator (GCD)** is equal to one. Hence, when **GCD**(a,N) = 1 the probing function **P**(x) be able to generate a complete cycle and we will always be able to find an empty bucket!

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	∅	∅	∅	∅	∅	∅	∅

Suppose we have an originally empty hash table and we want to insert some (k_i,v_i) pairs with LP and we selected our hash table to have:

P(x) = 6x

Probing function: **P**$(x) = 6x$
Fixed table size: $N = 9$
Max load factor: $\alpha = 0.667$
Threshold before resize = $N * \alpha = 6$

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	∅	∅	∅	∅	∅	∅	∅

Q: Based on the selected probing function **P**(x) and the table size are we likely to eventually get an infinite loop while inserting?

Probing function: **P**(x) = 6x
Fixed table size: N = 9
Max load factor: α = 0.667
Threshold before resize = N ∗ α = 6

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	∅	∅	∅	∅	∅	∅	∅

Q: Based on the selected probing function **P**(x) and the table size are we likely to eventually get an infinite loop while inserting?

A: Yes! **GCD**(N,a) = **GCD**(9,6) = **3** is not 1!

Probing function: **P**(x) = 6x
Fixed table size: N = 9
Max load factor: α = 0.667
Threshold before resize = N ∗ α = 6

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	∅	∅	∅	∅	∅	∅	∅

Recall, $P(x) = 6x$

Operations:

insert(k_1,v_1) ←
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4)
insert(k_5,v_5)
insert(k_6,v_6)

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	∅	∅	∅	∅	∅	∅	∅

Recall, $P(x) = 6x$

Suppose $H(k_1) = 2$

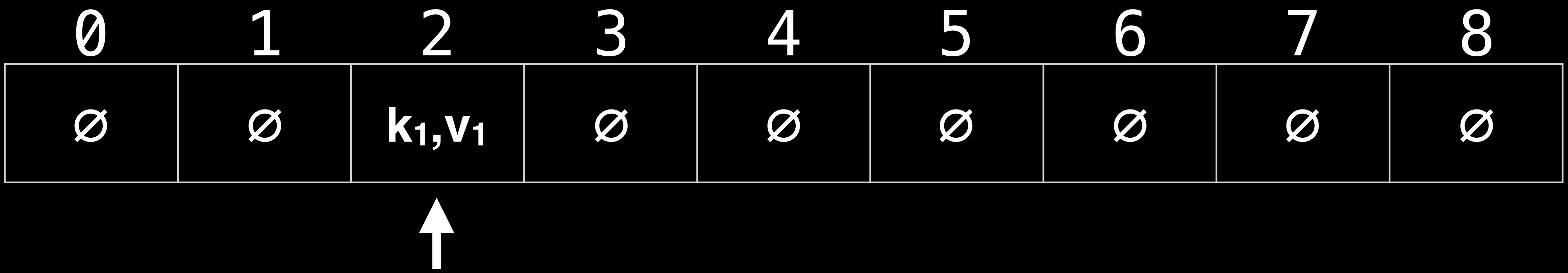
Operations:

insert(k_1,v_1) ←
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4)
insert(k_5,v_5)
insert(k_6,v_6)

$$(H(k_1) + P(0)) \bmod N =$$

$$(\quad 2 \quad + \quad 0\ ) \bmod 9 = 2$$

Inserting with LP



0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	∅	∅	∅	∅	∅	∅

Recall, $P(x) = 6x$

Suppose $H(k_1) = 2$

$(H(k_1) + P(0)) \bmod N =$

$(\quad 2 \quad + \quad 0 \ ) \bmod 9 = 2$

Operations:

insert(k_1,v_1) ←

insert(k_2,v_2)

insert(k_3,v_3)

insert(k_2,v_4)

insert(k_5,v_5)

insert(k_6,v_6)

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	**k_1,v_1**	∅	∅	∅	∅	∅	∅

Recall, **P**(x) = 6x

Operations:

insert(k_1,v_1)
insert(k_2,v_2) ←
insert(k_3,v_3)
insert(k_2,v_4)
insert(k_5,v_5)
insert(k_6,v_6)

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	∅	∅	∅	∅	∅	∅

Recall, $P(x) = 6x$

Suppose $H(k_2) = 2$

Operations:

insert(k_1,v_1)
insert(k_2,v_2) ←
insert(k_3,v_3)
insert(k_2,v_4)
insert(k_5,v_5)
insert(k_6,v_6)

$$(H(k_2) + P(0)) \bmod N =$$

$$(\quad 2 \quad + \quad 0\ ) \bmod 9 = 2$$

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	**k_1,v_1**	∅	∅	∅	∅	∅	∅

Operations:

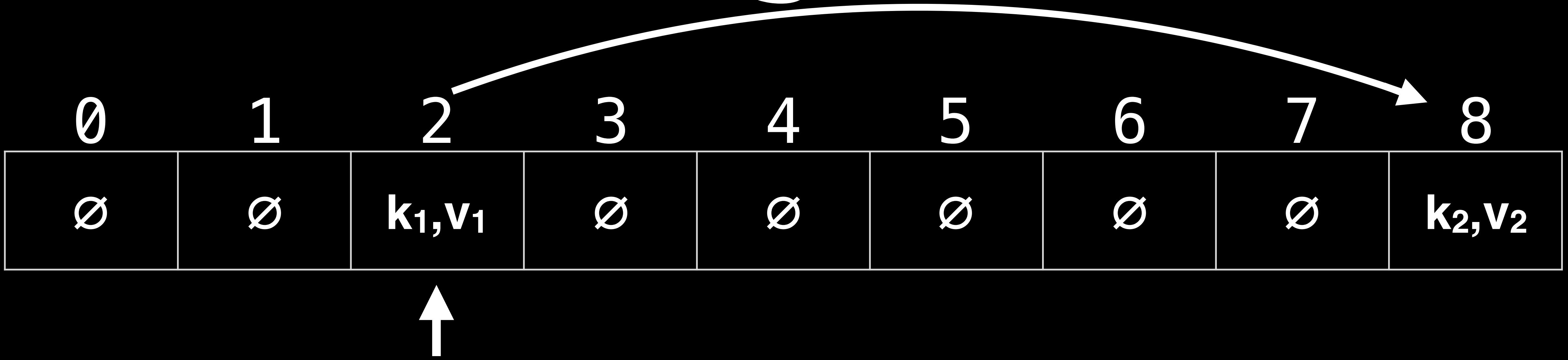
insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4)
insert(k_5,v_5)
insert(k_6,v_6)

Recall, **P**(x) = 6x
Suppose **H**(k_2) = 2

(**H**(k_2) + **P**(0)) mod N =
(2   +   0) mod 9 = 2

Hash collision! increment x and try offset **P**(1) instead of **P**(0)

Inserting with LP



0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	∅	∅	∅	∅	∅	k_2,v_2

Recall, $P(x) = 6x$

Suppose $H(k_2) = 2$

Operations:

$insert(k_1,v_1)$
$insert(k_2,v_2)$ ←
$insert(k_3,v_3)$
$insert(k_2,v_4)$
$insert(k_5,v_5)$
$insert(k_6,v_6)$

$(H(k_2) + P(0)) \text{ mod } N =$
$(\quad 2 \quad + \quad 0 \;) \text{ mod } 9 = 2$

$(H(k_2) + P(1)) \text{ mod } N =$
$(\quad 2 \quad + \quad 6 \;) \text{ mod } 9 = 8$

Hash collision! increment x and try offset $P(1)$ instead of $P(0)$

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	∅	∅	∅	∅	∅	k_2,v_2

Recall, **P**(x) = 6x

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3) ←
insert(k_2,v_4)
insert(k_5,v_5)
insert(k_6,v_6)

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	**k_1,v_1**	∅	∅	∅	∅	∅	**k_2,v_2**

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3) ←
insert(k_2,v_4)
insert(k_5,v_5)
insert(k_6,v_6)

Recall, $P(x) = 6x$

Suppose $H(k_3) = 3$

$$(H(k_3) + P(0)) \bmod N =$$

$$(\quad 3 \quad + \quad 0 \ ) \bmod 9 = 3$$

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	∅	∅	∅	k_2,v_2

Recall, $P(x) = 6x$

Suppose $H(k_3) = 3$

$(H(k_3) + P(0)) \bmod N =$

$(\ 3 + 0\ ) \bmod 9 = 3$

Operations:

$\text{insert}(k_1,v_1)$

$\text{insert}(k_2,v_2)$

$\text{insert}(k_3,v_3)$ ←

$\text{insert}(k_2,v_4)$

$\text{insert}(k_5,v_5)$

$\text{insert}(k_6,v_6)$

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	∅	∅	∅	k_2,v_2

Recall, **P**(x) = 6x

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4) ←
insert(k_5,v_5)
insert(k_6,v_6)

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	∅	∅	∅	k_2,v_2

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4) ←
insert(k_5,v_5)
insert(k_6,v_6)

Recall, **P**(x) = 6x

Notice that the key k_2 is already within the hash table, so instead of inserting we are **updating**. It's the same procedure except we update the value in the bucket when we find the key.

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	∅	∅	∅	k_2,v_2

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4) ←
insert(k_5,v_5)
insert(k_6,v_6)

Recall, $P(x) = 6x$

From before, $H(k_2) = 2$

$(H(k_2) + P(0)) \bmod N =$

$(\;\;2\;\;+\;\;0\;\;) \bmod 9 = 2$

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	∅	∅	∅	k_2,v_2
		↑						

Recall, $P(x) = 6x$

From before, $H(k_2) = 2$

$(H(k_2) + P(0)) \bmod N =$

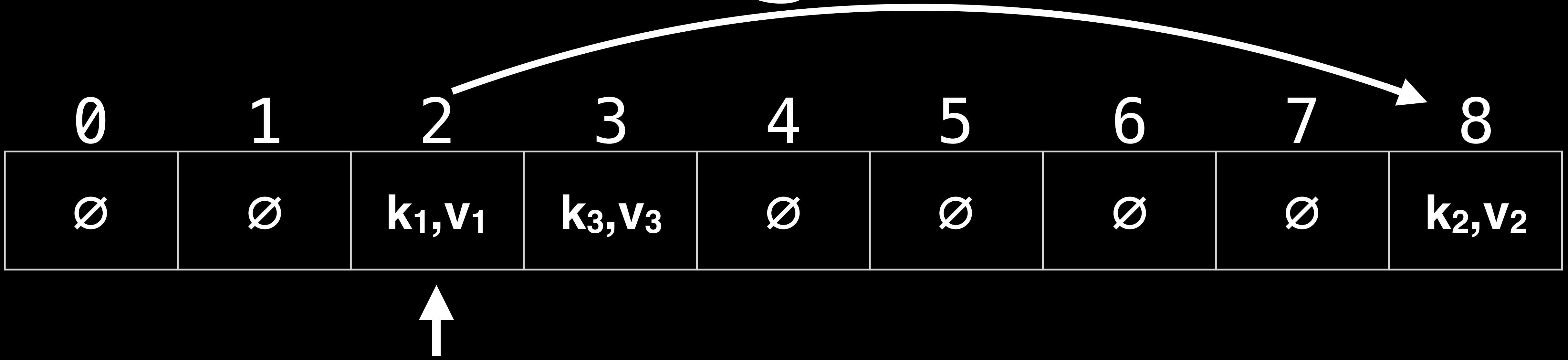
$(\ 2 + 0\ ) \bmod 9 = 2$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4) ←
insert(k_5,v_5)
insert(k_6,v_6)

Hash collision! increment x and try offset $P(1)$ instead of $P(0)$

Inserting with LP



0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	∅	∅	∅	k_2,v_2

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4) ←
insert(k_5,v_5)
insert(k_6,v_6)

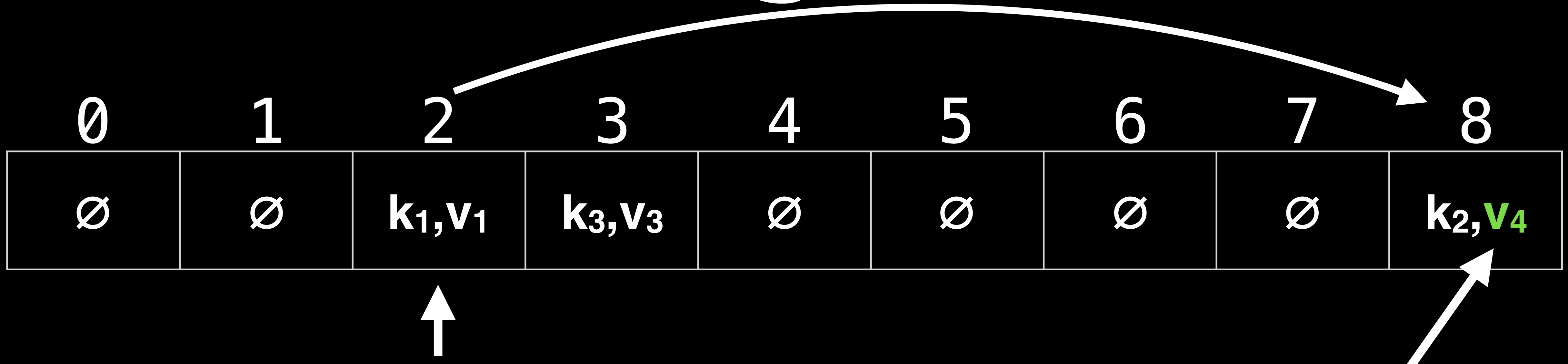
Recall, $P(x) = 6x$
From before, $H(k_2) = 2$

$$(H(k_2) + P(0)) \bmod N =$$
$$(\quad 2 \quad + \quad 0 \ ) \bmod 9 = 2$$

$$(H(k_2) + P(1)) \bmod N =$$
$$(\quad 2 \quad + \quad 6 \ ) \bmod 9 = 8$$

Hash collision! increment x and try offset $P(1)$ instead of $P(0)$

Inserting with LP



0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	∅	∅	∅	k_2,v_4

Recall, $P(x) = 6x$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4) ←
insert(k_5,v_5)
insert(k_6,v_6)

Update value to v_4

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	∅	∅	∅	k_2,v_4

Recall, **P**(x) = 6x

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4)
insert(k_5,v_5) ←
insert(k_6,v_6)

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	∅	∅	∅	k_2,v_4

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4)
insert(k_5,v_5) ←
insert(k_6,v_6)

Recall, $P(x) = 6x$

Suppose $H(k_5) = 8$

$$(H(k_5) + P(0)) \bmod N =$$

$$(\ 8 + 0\ ) \bmod 9 = 8$$

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	∅	∅	∅	k_2,v_4

Recall, $P(x) = 6x$

Suppose $H(k_5) = 8$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4)
insert(k_5,v_5) ←
insert(k_6,v_6)

$$(H(k_5) + P(0)) \bmod N =$$
$$(\ 8 + 0\ ) \bmod 9 = 8$$

Hash collision! increment x and try offset $P(1)$ instead of $P(0)$

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	∅	∅	∅	k_2,v_4

↑

Recall, $P(x) = 6x$

Suppose $H(k_5) = 8$

$$(H(k_5) + P(0)) \bmod N =$$

$$(\;8 + 0\;) \bmod 9 = 8$$

$$(H(k_5) + P(1)) \bmod N =$$

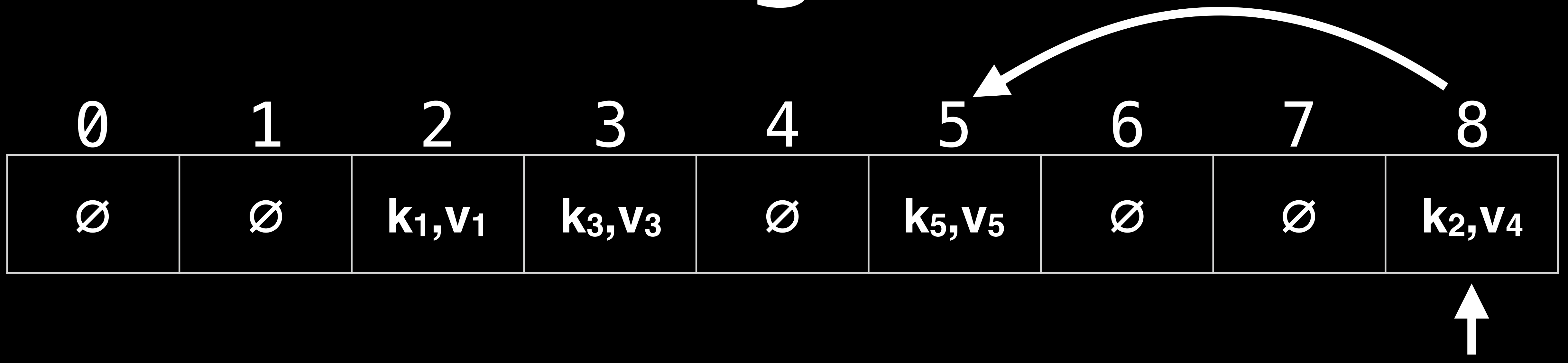
$$(\;8 + 6\;) \bmod 9 = 5$$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4)
insert(k_5,v_5) ←
insert(k_6,v_6)

Hash collision! increment x and try offset P(1) instead of P(0)

Inserting with LP



0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	k_5,v_5	∅	∅	k_2,v_4

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4)
insert(k_5,v_5) ←
insert(k_6,v_6)

Recall, $P(x) = 6x$
Suppose $H(k_5) = 8$

$$(H(k_5) + P(0)) \bmod N =$$
$$(\quad 8 \quad + \quad 0\ ) \bmod 9 = 8$$
$$(H(k_5) + P(1)) \bmod N =$$
$$(\quad 8 \quad + \quad 6\ ) \bmod 9 = 5$$

Hash collision! increment x and try offset P(1) instead of P(0)

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	k_5,v_5	∅	∅	k_2,v_4

Recall, $P(x) = 6x$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4)
insert(k_5,v_5)
insert(k_6,v_6) ←

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	k_5,v_5	∅	∅	k_2,v_4

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4)
insert(k_5,v_5)
insert(k_6,v_6) ←

Recall, $P(x) = 6x$

Suppose $H(k_6) = 5$

$$(H(k_6) + P(0)) \bmod N =$$

$$(\quad 5 \quad + \quad 0 \;) \bmod 9 = 5$$

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	k_5,v_5	∅	∅	k_2,v_4
					↑			

Operations:

$insert(k_1,v_1)$
$insert(k_2,v_2)$
$insert(k_3,v_3)$
$insert(k_2,v_4)$
$insert(k_5,v_5)$
$insert(k_6,v_6)$ ←

Recall, $P(x) = 6x$
Suppose $H(k_6) = 5$

$(H(k_6) + P(0)) \bmod N =$
$(\quad 5 \quad + \quad 0 \;) \bmod 9 = 5$

Inserting with LP

0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	k_5,v_5	∅	∅	k_2,v_4

↑

Recall, $P(x) = 6x$

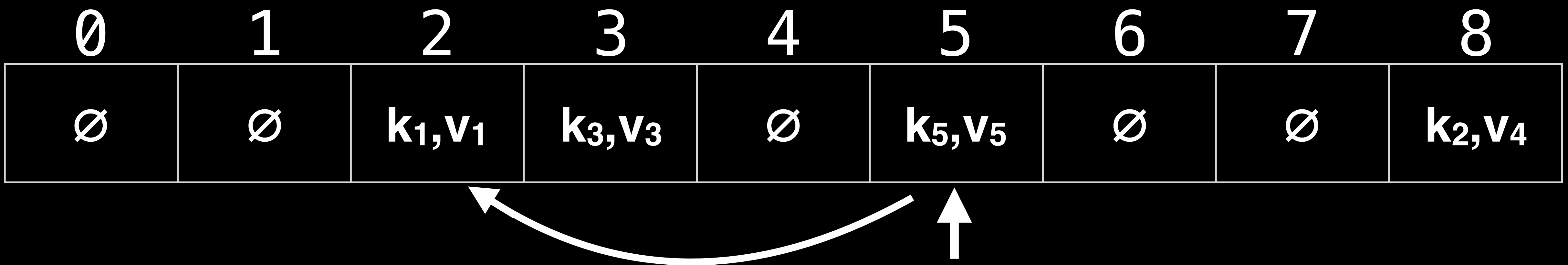
Suppose $H(k_6) = 5$

$$(H(k_6) + P(0)) \bmod N =$$
$$(\;5 \;+\; 0\;) \bmod 9 = 5$$
$$(H(k_6) + P(1)) \bmod N =$$
$$(\;5 \;+\; 6\;) \bmod 9 = 2$$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4)
insert(k_5,v_5)
insert(k_6,v_6) ←

Inserting with LP



0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	k_5,v_5	∅	∅	k_2,v_4

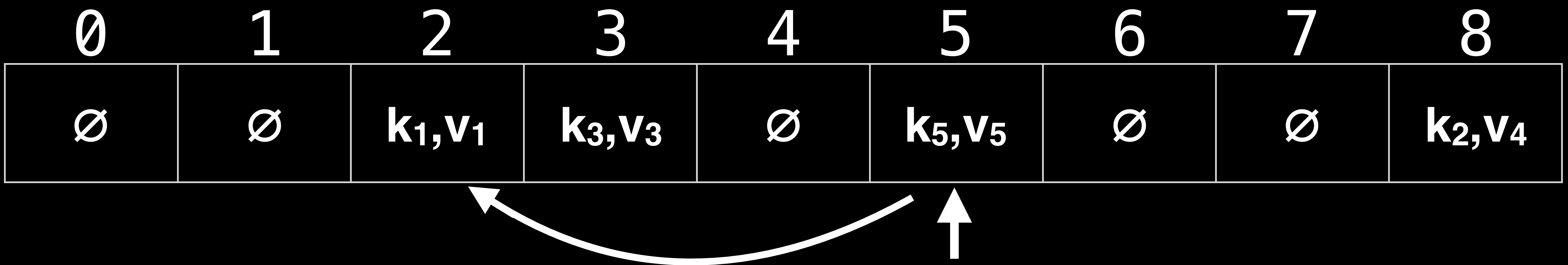
Operations:

$\text{insert}(k_1,v_1)$
$\text{insert}(k_2,v_2)$
$\text{insert}(k_3,v_3)$
$\text{insert}(k_2,v_4)$
$\text{insert}(k_5,v_5)$
$\text{insert}(k_6,v_6) \leftarrow$

Recall, $P(x) = 6x$
Suppose $H(k_6) = 5$

$(H(k_6) + P(0)) \bmod N =$
$(\;5\;+\;0\;) \bmod 9 = 5$
$(H(k_6) + P(1)) \bmod N =$
$(\;5\;+\;6\;) \bmod 9 = 2$

Inserting with LP



0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	k_5,v_5	∅	∅	k_2,v_4

Recall, $P(x) = 6x$

Suppose $H(k_6) = 5$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4)
insert(k_5,v_5)
insert(k_6,v_6) ←

$(H(k_6) + P(0)) \bmod N =$

$(\quad 5 \quad + \quad 0 \ ) \bmod 9 = 5$

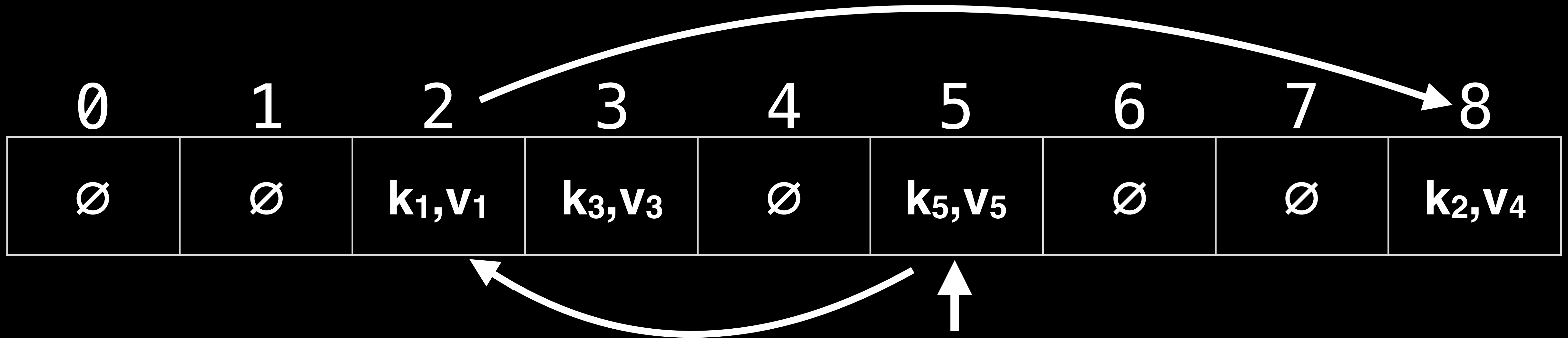
$(H(k_6) + P(1)) \bmod N =$

$(\quad 5 \quad + \quad 6 \ ) \bmod 9 = 2$

$(H(k_6) + P(2)) \bmod N =$

$(\quad 5 \quad + 12 \ ) \bmod 9 = 8$

Inserting with LP



0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	k_5,v_5	∅	∅	k_2,v_4

Recall, $P(x) = 6x$

Suppose $H(k_6) = 5$

$(H(k_6) + P(0)) \bmod N =$

$(\quad 5 \quad + \quad 0 \quad) \bmod 9 = 5$

$(H(k_6) + P(1)) \bmod N =$

$(\quad 5 \quad + \quad 6 \quad) \bmod 9 = 2$

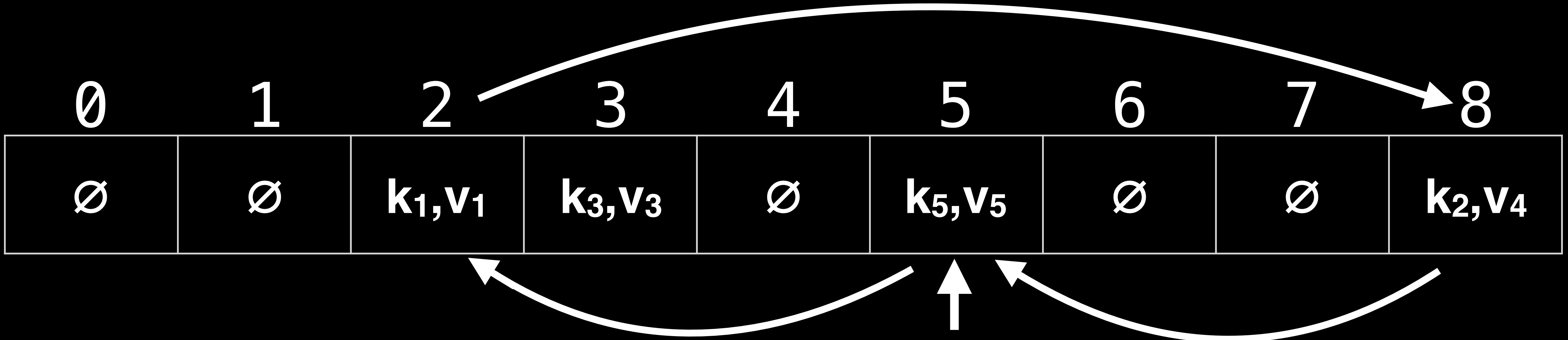
$(H(k_6) + P(2)) \bmod N =$

$(\quad 5 \quad + 12 \quad) \bmod 9 = 8$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_2,v_4)
insert(k_5,v_5)
insert(k_6,v_6) ←

Inserting with LP



0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	k_5,v_5	∅	∅	k_2,v_4

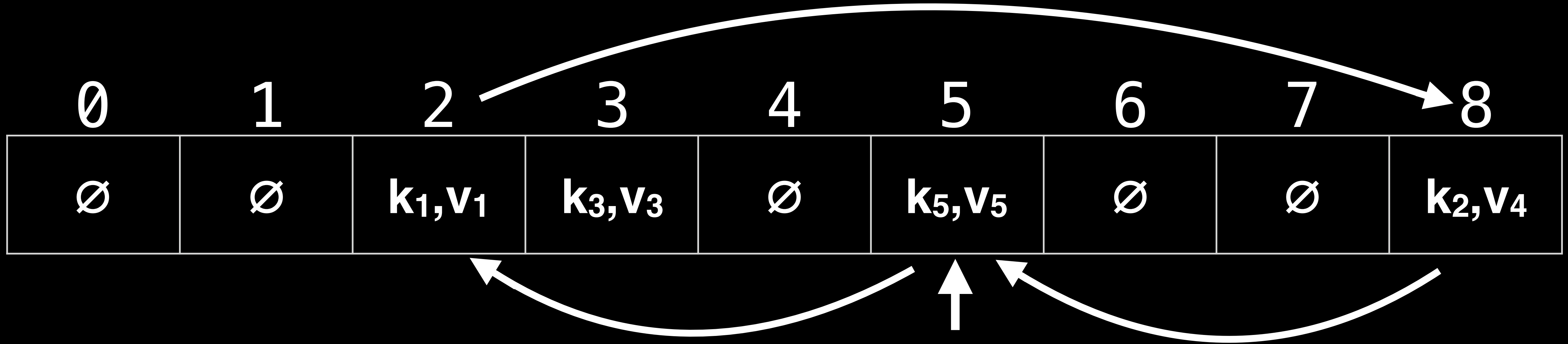
Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_5,v_5)
insert(k_6,v_6) ←

Recall, $P(x) = 6x$
Suppose $H(k_6) = 5$

$(H(k_6) + P(0)) \bmod N =$
$(\quad 5 \quad + \quad 0 \quad) \bmod 9 = 5$
$(H(k_6) + P(1)) \bmod N =$
$(\quad 5 \quad + \quad 6 \quad) \bmod 9 = 2$
$(H(k_6) + P(2)) \bmod N =$
$(\quad 5 \quad + 12 \quad) \bmod 9 = 8$
$(H(k_6) + P(3)) \bmod N =$
$(\quad 5 \quad + 18 \quad) \bmod 9 = 5$

Inserting with LP

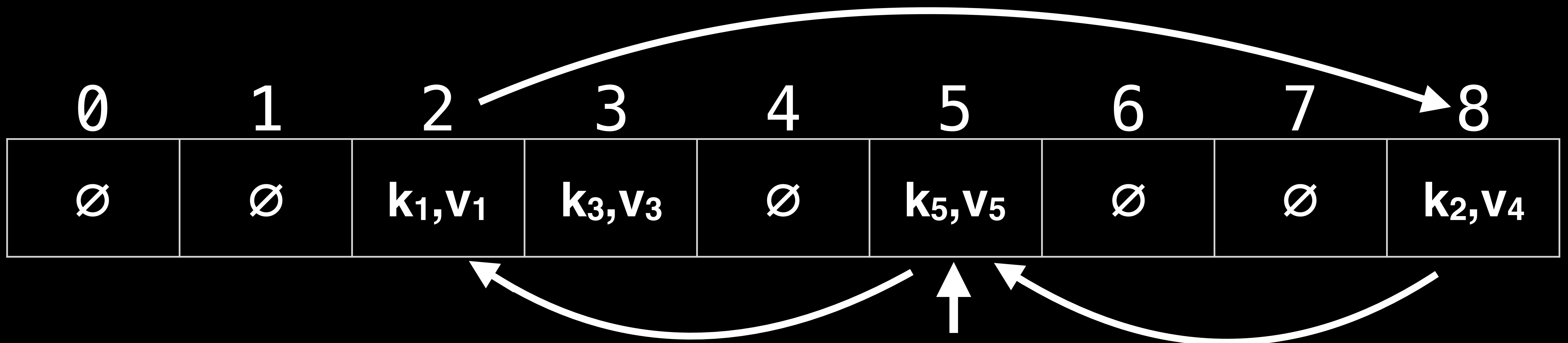


0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	k_5,v_5	∅	∅	k_2,v_4

Recall, **P**(x) = 6x

Oh no, we're trapped in a cycle! However, we expected this to happen since **GCD**(9,6) = 3

Inserting with LP



0	1	2	3	4	5	6	7	8
∅	∅	k_1,v_1	k_3,v_3	∅	k_5,v_5	∅	∅	k_2,v_4

Recall, P(x) = 6x

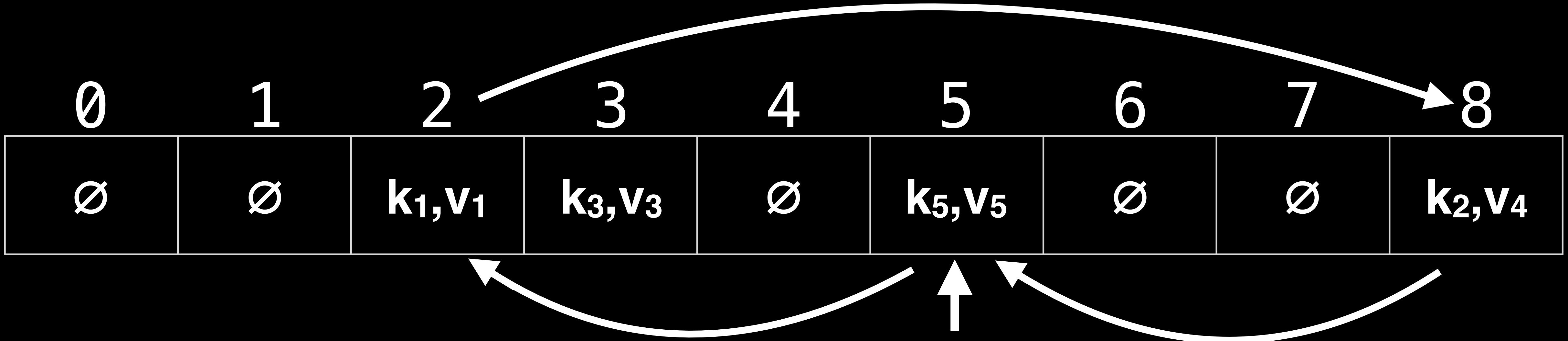
Oh no, we’re trapped in a cycle! However, we expected this to happen since GCD(9,6) = 3

GCD(9,1) = 1
GCD(9,2) = 1
GCD(9,3) = 3

GCD(9,4) = 1
GCD(9,5) = 1
GCD(9,6) = 3

GCD(9,7) = 1
GCD(9,8) = 1
GCD(9,9) = 9

Inserting with LP



Recall, P(x) = 6x

Oh no, we’re trapped in a cycle! However, we expected this to happen since GCD(9,6) = 3

GCD(9,1) = 1   GCD(9,4) = 1   GCD(9,7) = 1
GCD(9,2) = 1   GCD(9,5) = 1   GCD(9,8) = 1
GCD(9,3) = 3   GCD(9,6) = 3   GCD(9,9) = 9

A common choice for P(x) is P(x) = 1x since GCD(N,1) = 1 no matter the choice of N (table size)

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

Suppose we have an originally empty hash table and we want to insert some (k_i,v_i) pairs with LP and we selected our hash table to have:

Probing function: **P**(x) = 5x
Fixed table size: N = 12
Max load factor: α = 0.35
Threshold before resize = N * α = 4

GCD(12,5) = 1 so no cycle should occur!

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

Recall $P(x) = 5x$, $N = 12$, threshold = 4

Operations:

insert(k_1,v_1) ←

insert(k_2,v_2)

insert(k_3,v_3)

insert(k_4,v_4)

insert(k_5,v_5)

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

Recall $P(x) = 5x$, $N = 12$, threshold = 4

Operations:

insert(k_1,v_1) ←

insert(k_2,v_2)

insert(k_3,v_3)

insert(k_4,v_4)

insert(k_5,v_5)

Suppose $H(k_1) = 10$

$H(k_1) + P(0) \bmod N = 10$

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	k_1,v_1	∅

↑

Recall $P(x) = 5x$, $N = 12$, threshold $= 4$

Operations:

insert(k_1,v_1) ←

insert(k_2,v_2)

insert(k_3,v_3)

insert(k_4,v_4)

insert(k_5,v_5)

Suppose $H(k_1) = 10$

$H(k_1) + P(0) \bmod N = 10$

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	k_1,v_1	∅

Recall **P**(x) = 5x, N = 12, threshold = 4

Operations:

insert(k_1,v_1)
insert(k_2,v_2) ←
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_5,v_5)

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	k_1, v_1	∅

Recall $P(x) = 5x$, $N = 12$, threshold = 4

Operations:

insert(k_1,v_1)
insert(k_2,v_2) ←
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_5,v_5)

Suppose $H(k_2) = 8$

$H(k_2) + P(0) \bmod N = 8$

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

Recall $P(x) = 5x$, $N = 12$, threshold = 4

Operations:

insert(k_1,v_1)
insert(k_2,v_2) ←
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_5,v_5)

Suppose $H(k_2) = 8$

$H(k_2) + P(0) \bmod N = 8$

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

Recall **P**(x) = 5x, N = 12, threshold = 4

Operations:

insert(k_1,v_1)

insert(k_2,v_2)

insert(k_3,v_3) ←

insert(k_4,v_4)

insert(k_5,v_5)

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

Recall $P(x) = 5x$, $N = 12$, threshold = 4

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3) ←
insert(k_4,v_4)
insert(k_5,v_5)

Suppose $H(k_3) = 10$

$H(k_3) + P(0) \bmod N = 10$

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

Recall $P(x) = 5x$, $N = 12$, threshold = 4

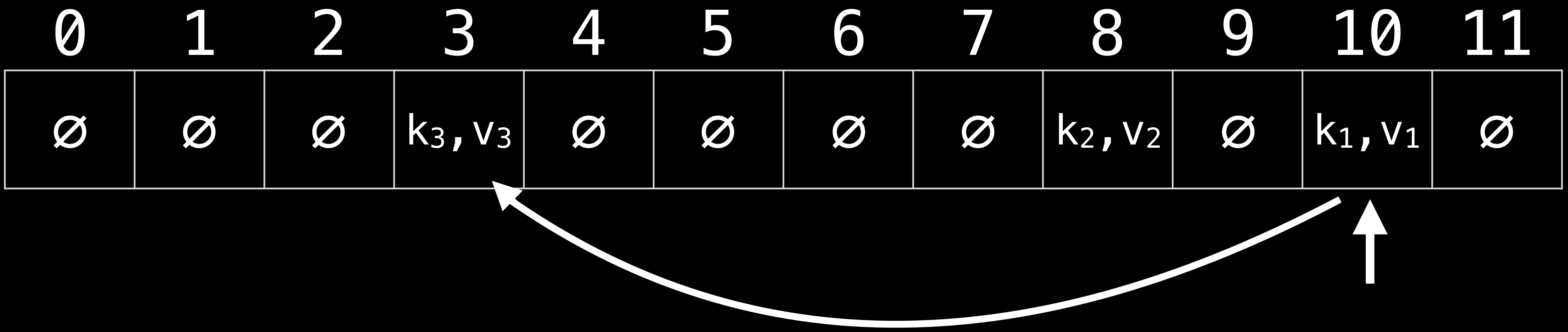
Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3) ←
insert(k_4,v_4)
insert(k_5,v_5)

Suppose $H(k_3) = 10$

$H(k_3) + P(0) \bmod N = 10$

Oops cell 10 is already taken so keep probing



0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	k_3, v_3	∅	∅	∅	∅	k_2, v_2	∅	k_1, v_1	∅

Recall $P(x) = 5x$, $N = 12$, threshold $= 4$

Operations:

insert(k_1,v_1)

insert(k_2,v_2)

insert(k_3,v_3) ←

insert(k_4,v_4)

insert(k_5,v_5)

Suppose $H(k_3) = 10$

$H(k_3) + P(0) \bmod N = 10$

$H(k_3) + P(1) \bmod N = 3$

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

Recall **P**(x) = 5x, N = 12, threshold = 4

Operations:

insert(k_1,v_1)

insert(k_2,v_2)

insert(k_3,v_3)

insert(k_4,v_4) ←

insert(k_5,v_5)

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

Recall $P(x) = 5x$, $N = 12$, threshold = 4

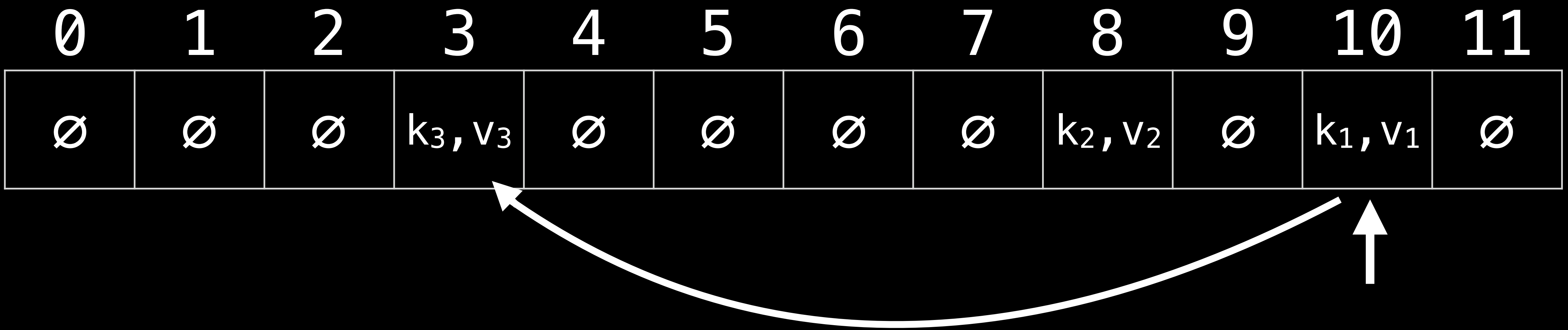
Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_5,v_5)

Suppose $H(k_4) = 10$

$H(k_4) + P(0) \text{ mod } N = 10$

Oops cell 10 is already taken so keep probing



0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

Recall **P**(x) = 5x, N = 12, threshold = 4

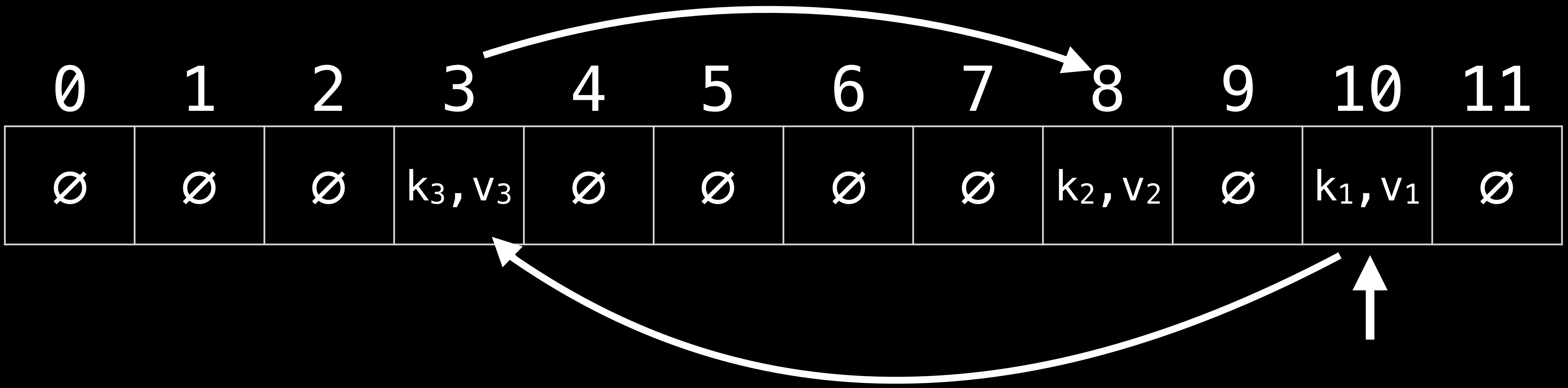
Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4) ←
insert(k_5,v_5)

Suppose **H**(k_4) = 10

H(k_4) + **P**(0) mod N = 10
H(k_4) + **P**(1) mod N = 3

Oops cell 3 is already taken so keep probing



0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

Recall **P**(x) = 5x, N = 12, threshold = 4

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4) ←
insert(k_5,v_5)

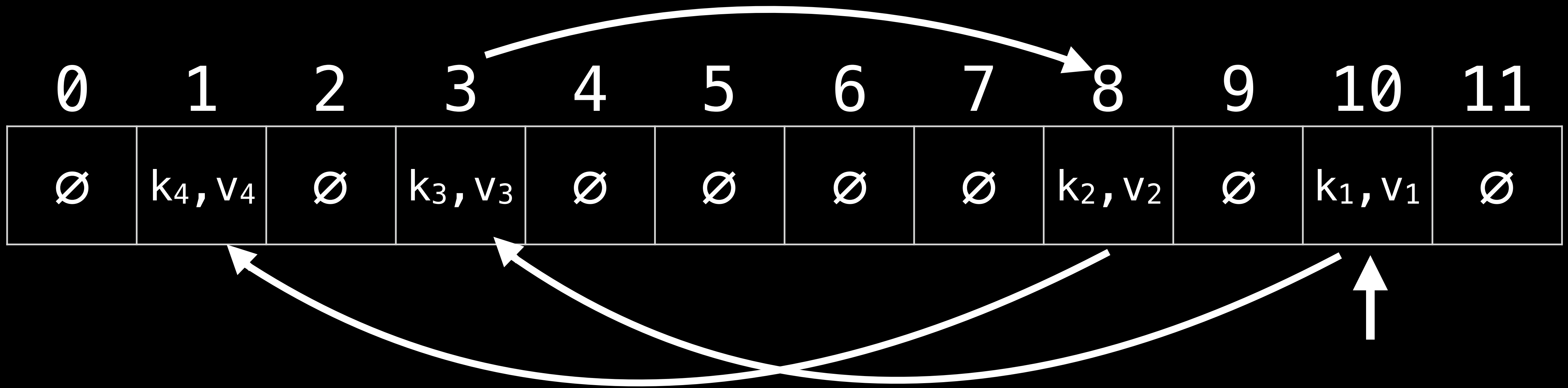
Suppose **H**(k_4) = 10

H(k_4) + **P**(0) mod N = 10

H(k_4) + **P**(1) mod N = 3

H(k_4) + **P**(2) mod N = 8

Oops cell 8 is already taken so keep probing



0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

Recall **P**(x) = 5x, N = 12, threshold = 4

Operations:

insert(k_1,v_1)

insert(k_2,v_2)

insert(k_3,v_3)

insert(k_4,v_4) ←

insert(k_5,v_5)

Suppose **H**(k_4) = 10

H(k_4) + **P**(0) mod N = 10

H(k_4) + **P**(1) mod N = 3

H(k_4) + **P**(2) mod N = 8

H(k_4) + **P**(3) mod N = 1

0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

Before we insert the next (k_i,v_i) pair, notice that we have reached the threshold value, so we need to grow the table. Usually this is done in some exponential fashion such as doubling the table size. Whatever you do make sure **GCD**(N,a) = 1 still holds.

After doubling N = 24
α is constant so it’s still 0.35
New threshold value = N * α = 8
The probing function **P**(x) does not change.

0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

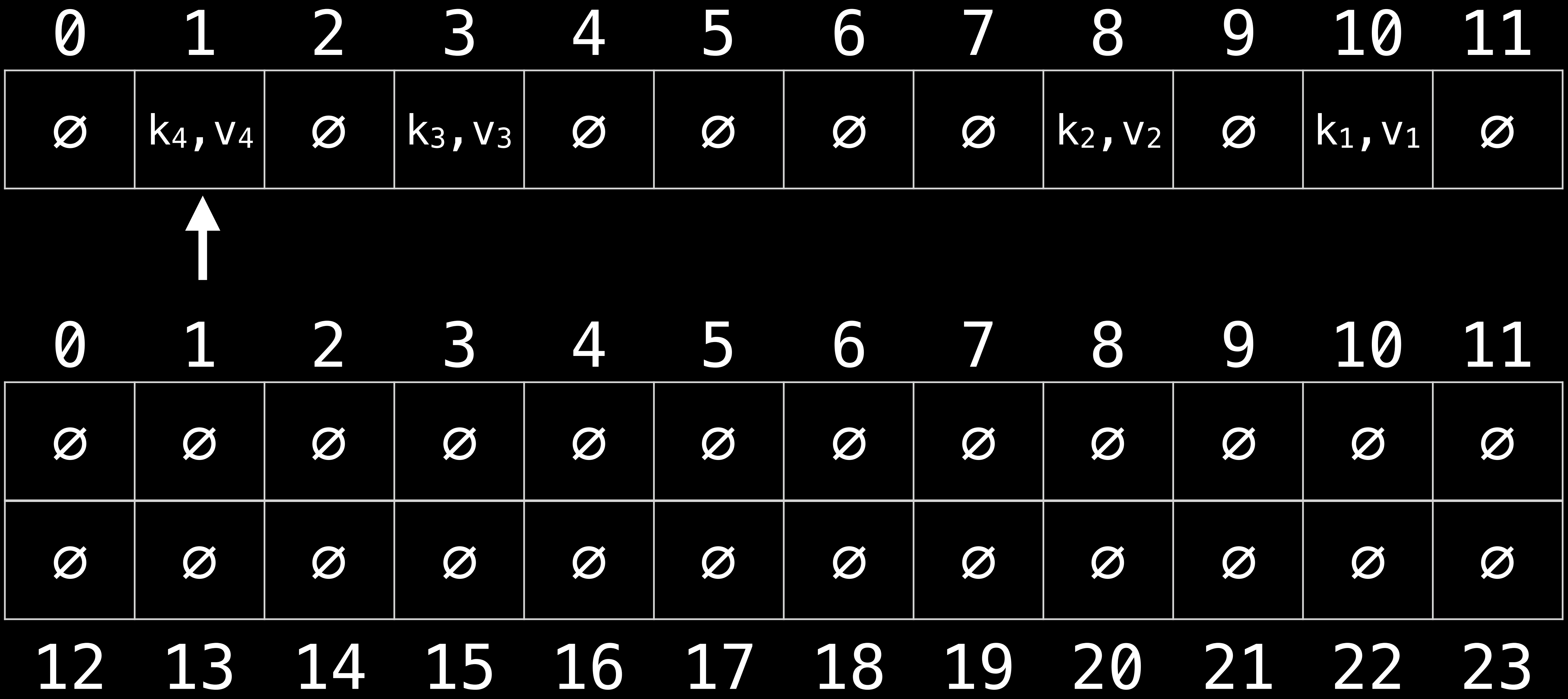
Upon allocating memory for a new table we need to insert the contents of the old table into the new table.

Recall **P**(x) = 5x, N = 24, threshold = 8

0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

Recall **P**(x) = 5x, N = 24, threshold = 8



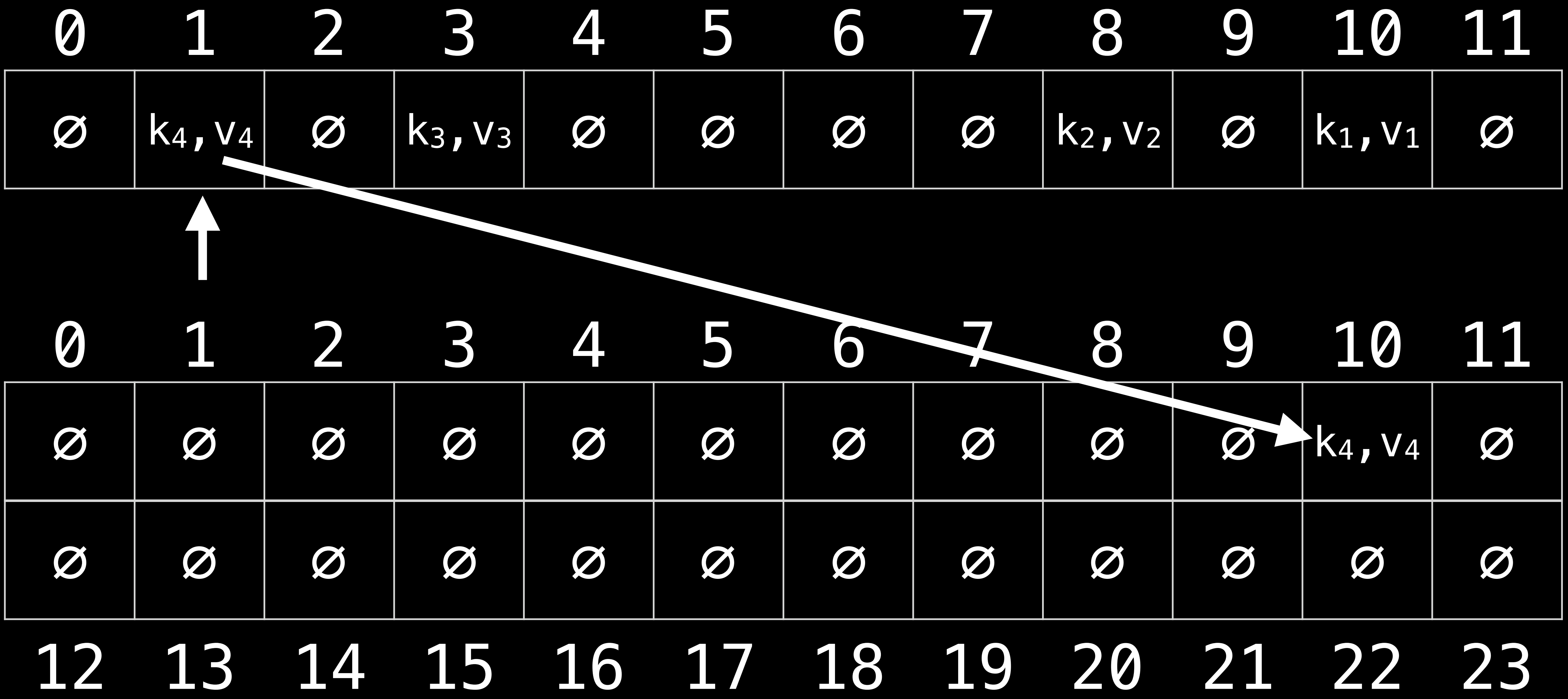
0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4, v_4	∅	k_3, v_3	∅	∅	∅	∅	k_2, v_2	∅	k_1, v_1	∅

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

From before, $H(k_4) = 10$

$H(k_4) + P(0) \bmod N = 10$

Recall $P(x) = 5x$, $N = 24$, threshold $= 8$



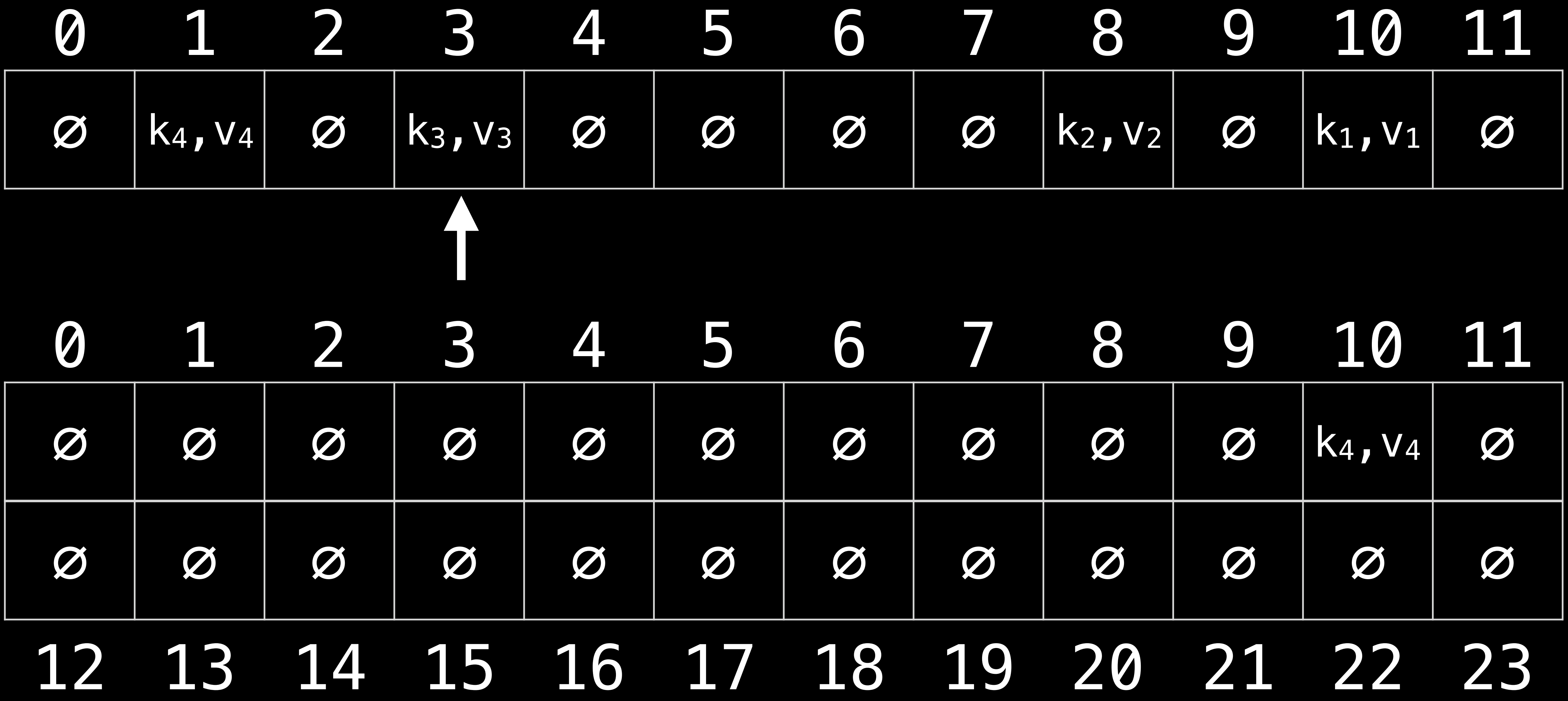
0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	k_4,v_4	∅
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

From before, $H(k_4) = 10$

$H(k_4) + P(0) \bmod N = 10$

Recall $P(x) = 5x$, $N = 24$, threshold = 8



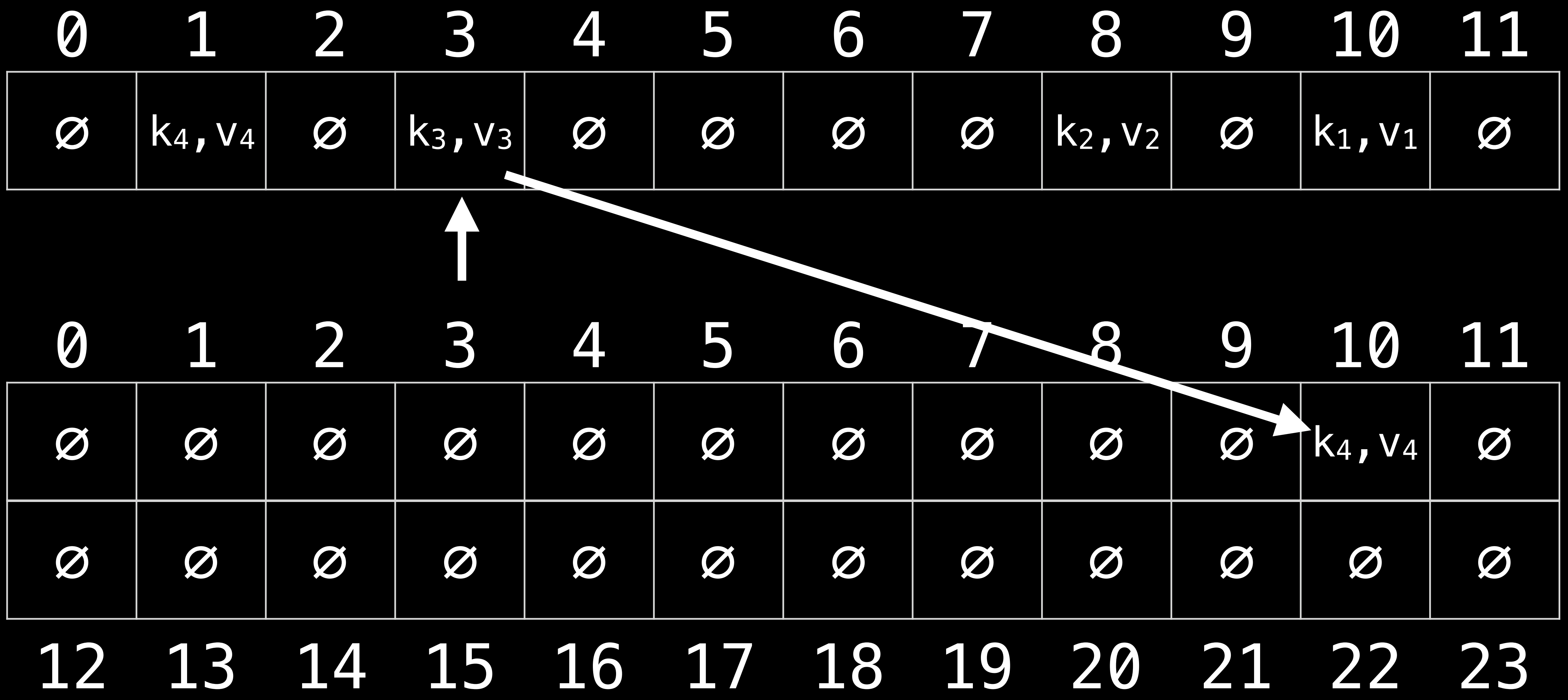
0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4, v_4	∅	k_3, v_3	∅	∅	∅	∅	k_2, v_2	∅	k_1, v_1	∅

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	k_4, v_4	∅
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

From before, $\mathbf{H}(k_3) = 10$

$\mathbf{H}(k_3) + \mathbf{P}(0) \bmod N = 10$

Recall $\mathbf{P}(x) = 5x$, $N = 24$, threshold $= 8$



0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

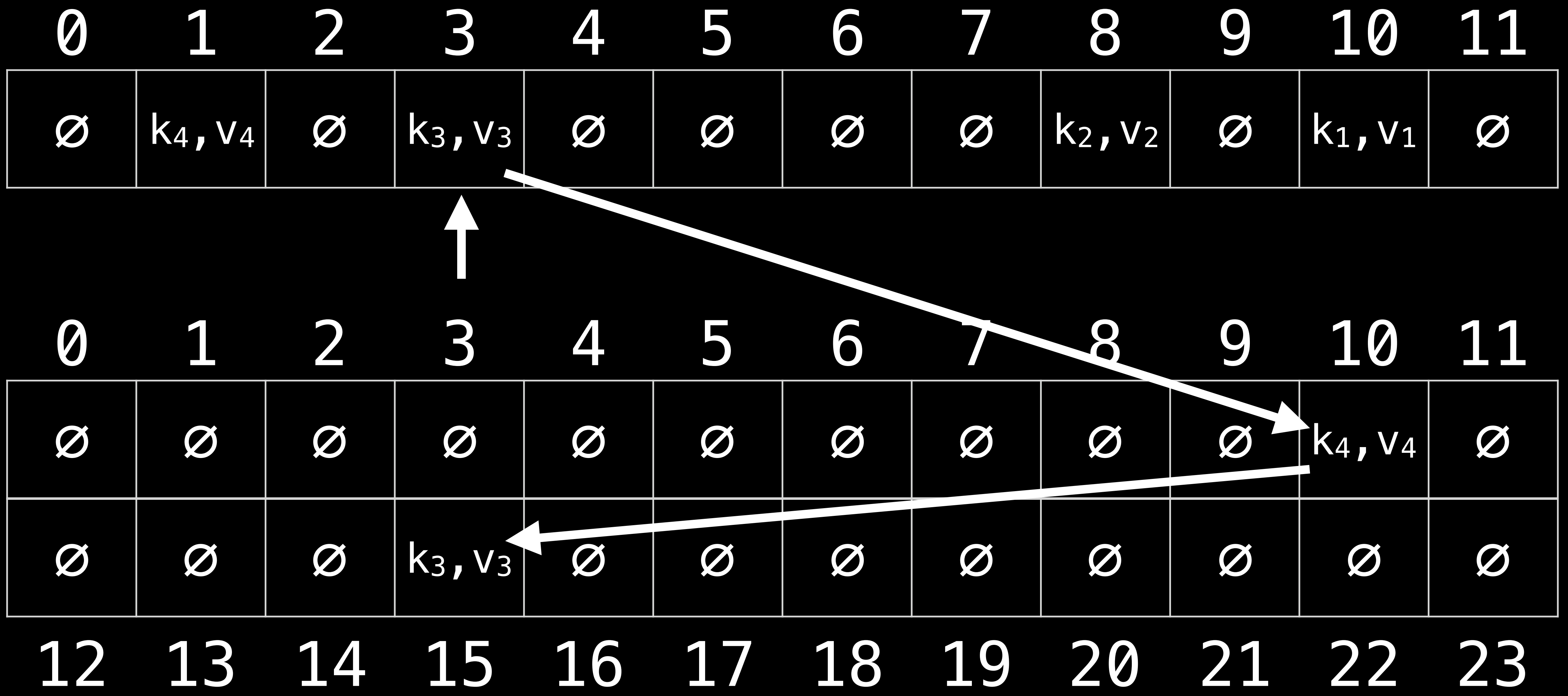
0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	k_4,v_4	∅
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

From before, $H(k_3) = 10$

$H(k_3) + P(0) \bmod N = 10$

There’s a collision at position 10 in the new table, so keep probing.

Recall $P(x) = 5x$, $N = 24$, threshold = 8



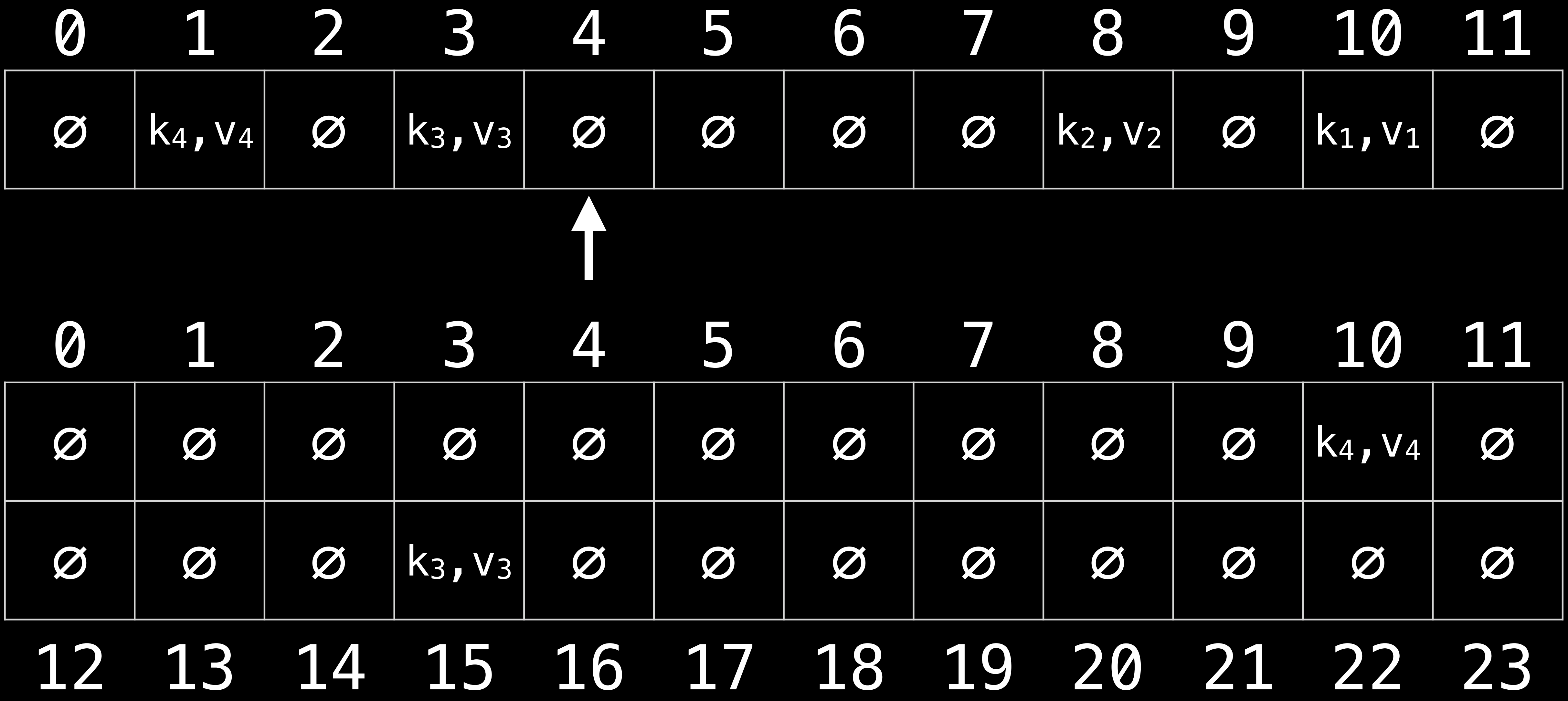
0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	k_4,v_4	∅
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

From before, $\mathbf{H}(k_3) = 10$

$\mathbf{H}(k_3) + \mathbf{P}(0) \bmod N = 10$
$\mathbf{H}(k_3) + \mathbf{P}(1) \bmod N = 15$

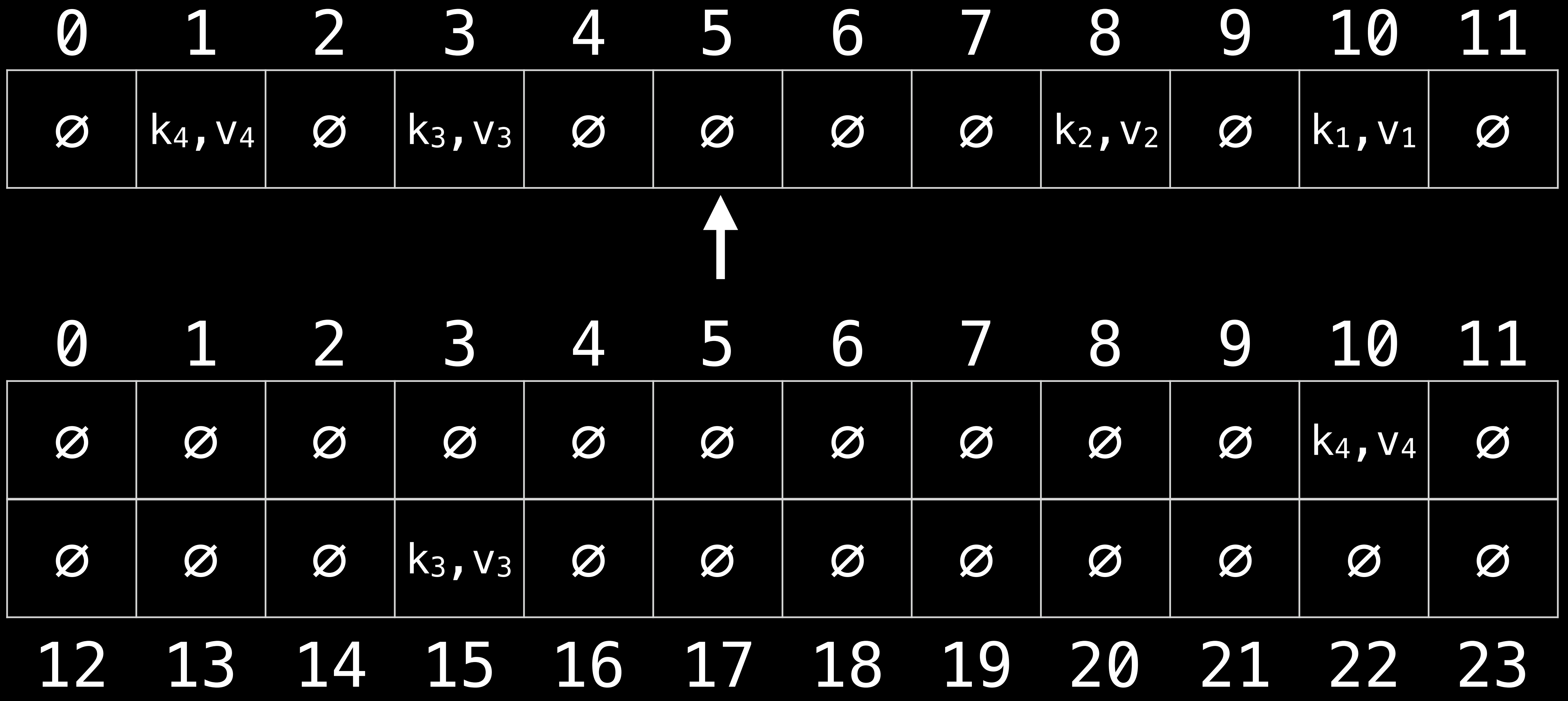
Recall $\mathbf{P}(x) = 5x$, $N = 24$, threshold = 8



0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	k_4,v_4	∅
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

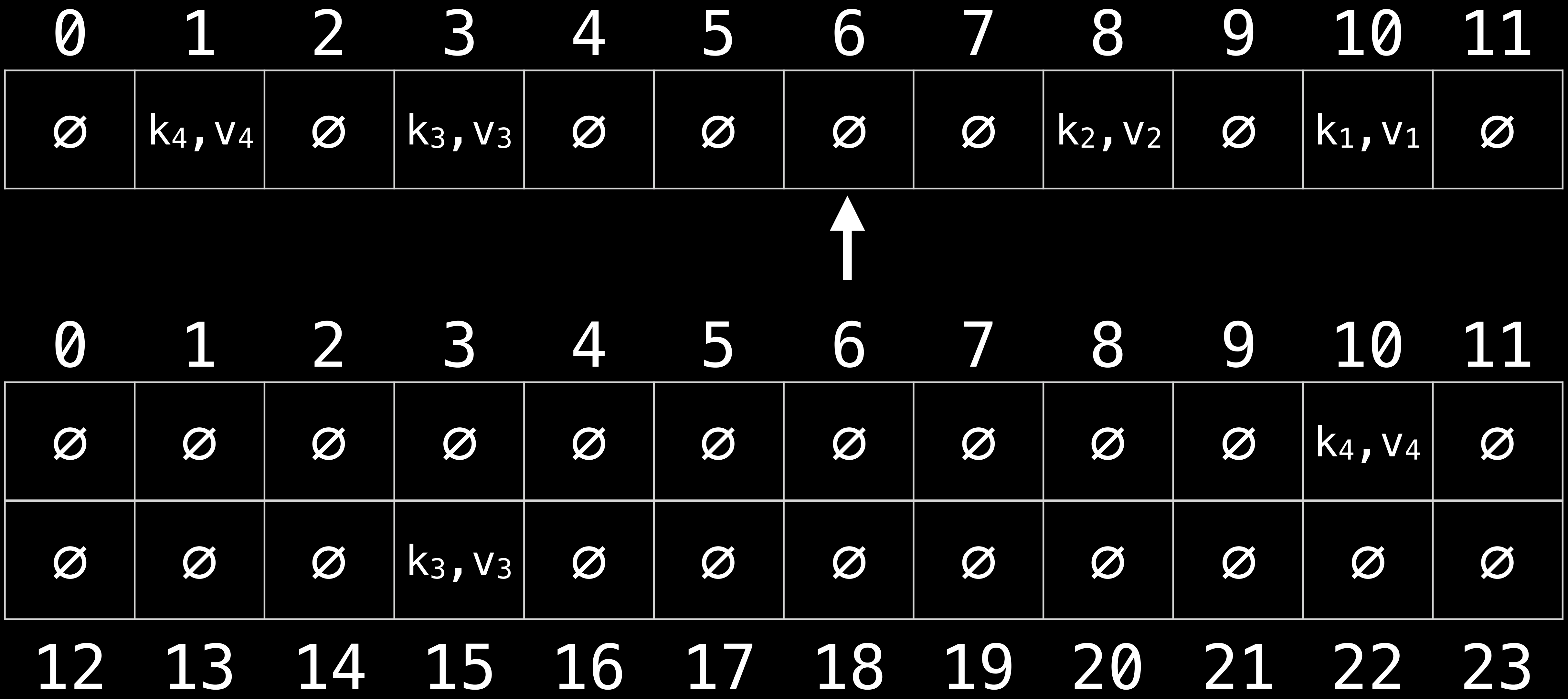
Recall P(x) = 5x, N = 24, threshold = 8



0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	k_4,v_4	∅
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

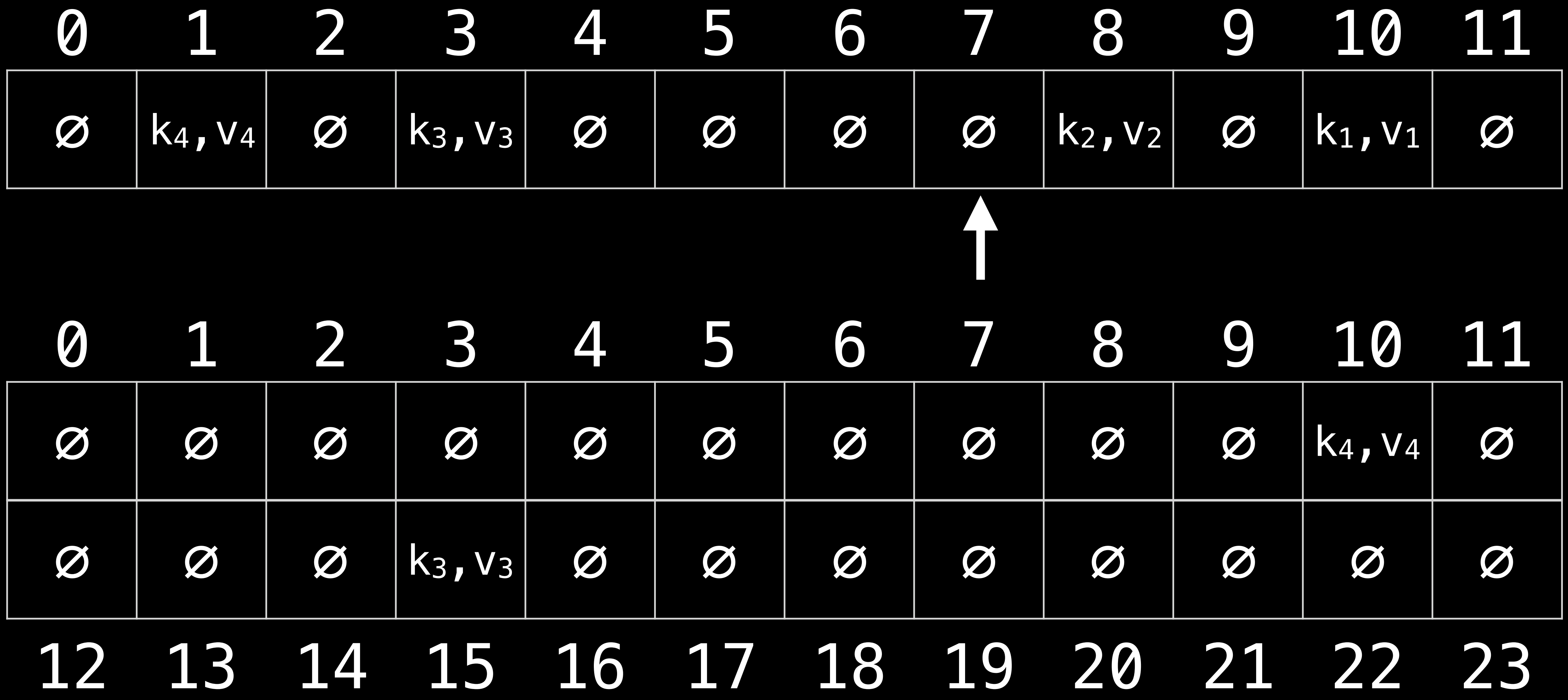
Recall **P**(x) = 5x, N = 24, threshold = 8



0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4, v_4	∅	k_3, v_3	∅	∅	∅	∅	k_2, v_2	∅	k_1, v_1	∅

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	k_4, v_4	∅
∅	∅	∅	k_3, v_3	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

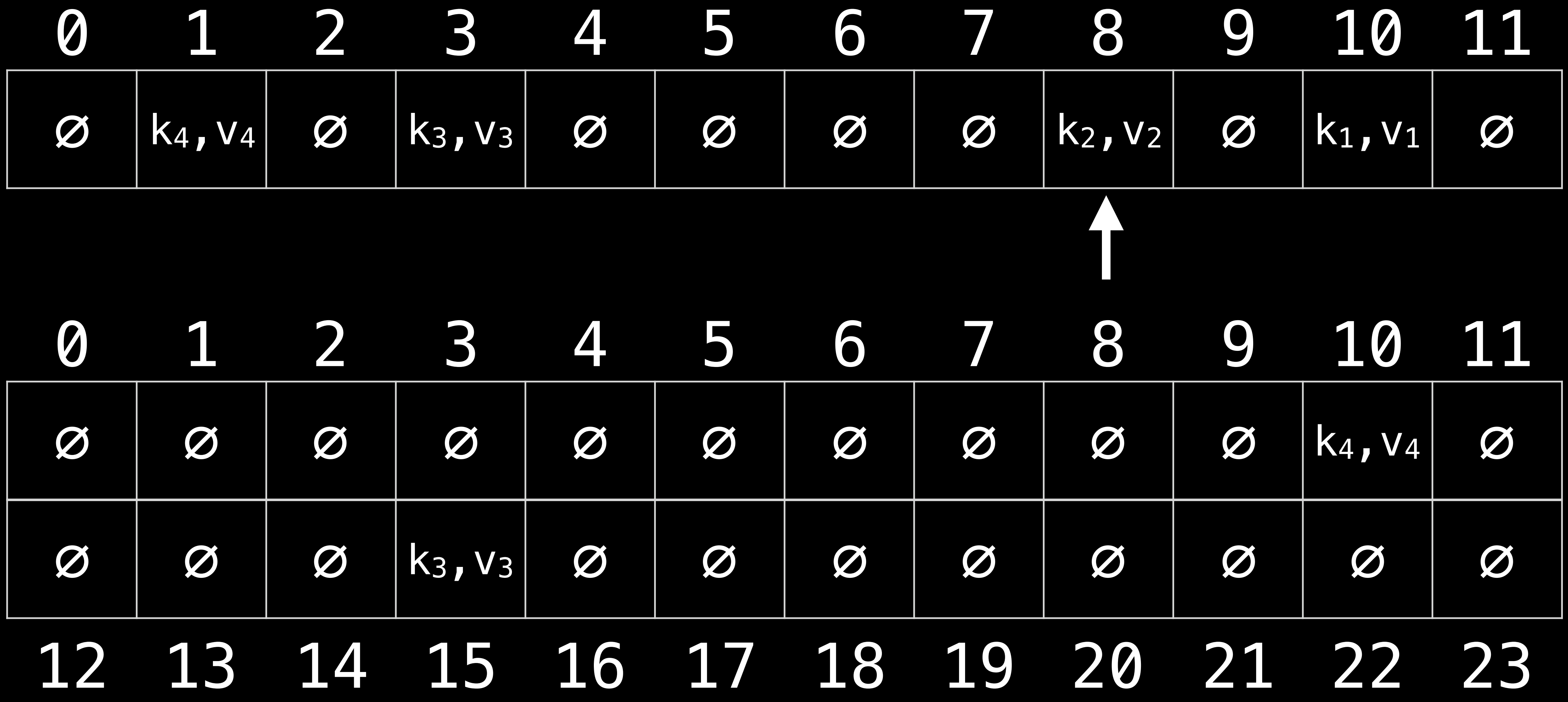
Recall **P**(x) = 5x, N = 24, threshold = 8



0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4, v_4	∅	k_3, v_3	∅	∅	∅	∅	k_2, v_2	∅	k_1, v_1	∅

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	k_4, v_4	∅
∅	∅	∅	k_3, v_3	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

Recall **P**(x) = 5x, N = 24, threshold = 8



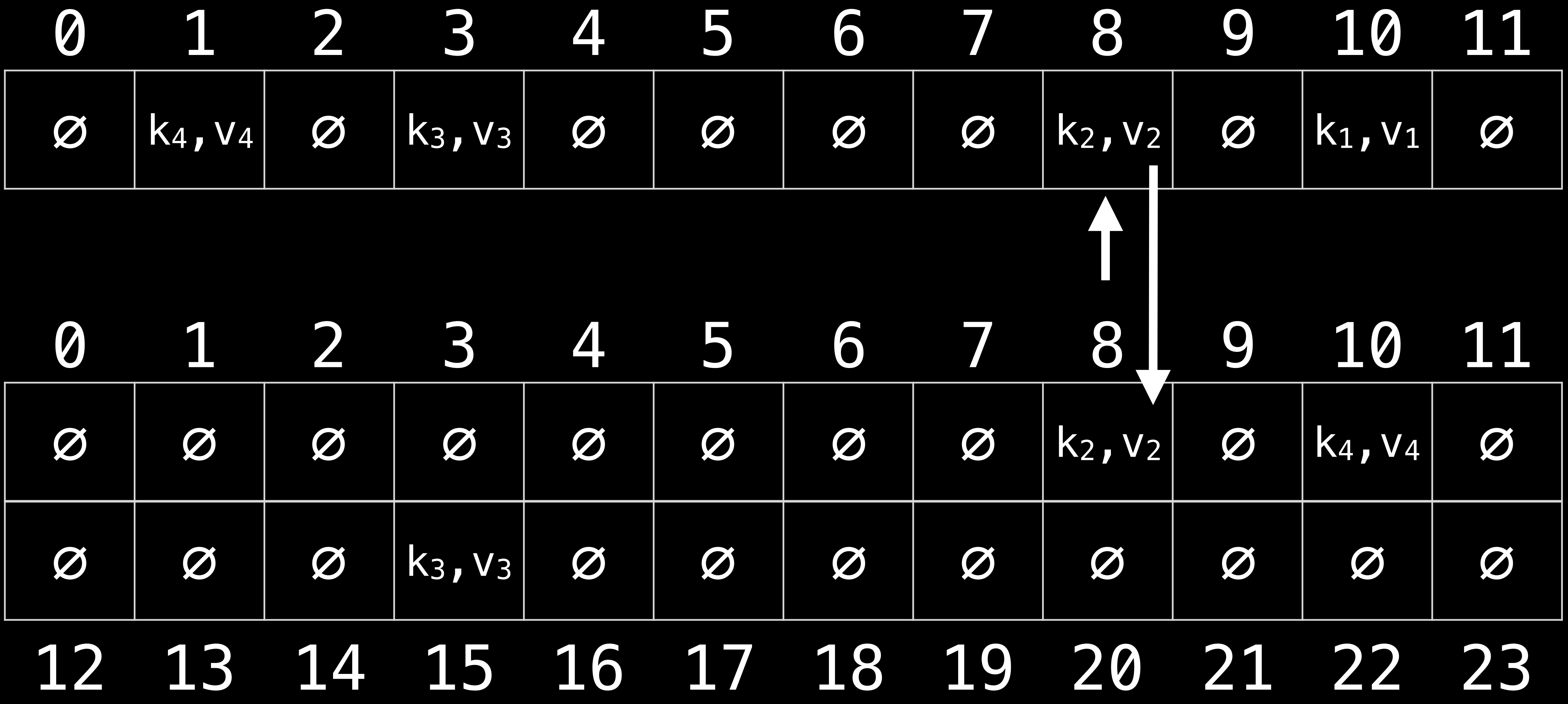
0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	k_4,v_4	∅
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

From before, $H(k_2) = 8$

$H(k_2) + P(0) \bmod N = 8$

Recall $P(x) = 5x$, $N = 24$, threshold $= 8$



0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	k_2,v_2	∅	k_4,v_4	∅
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

From before, $\mathbf{H}(k_2) = 8$

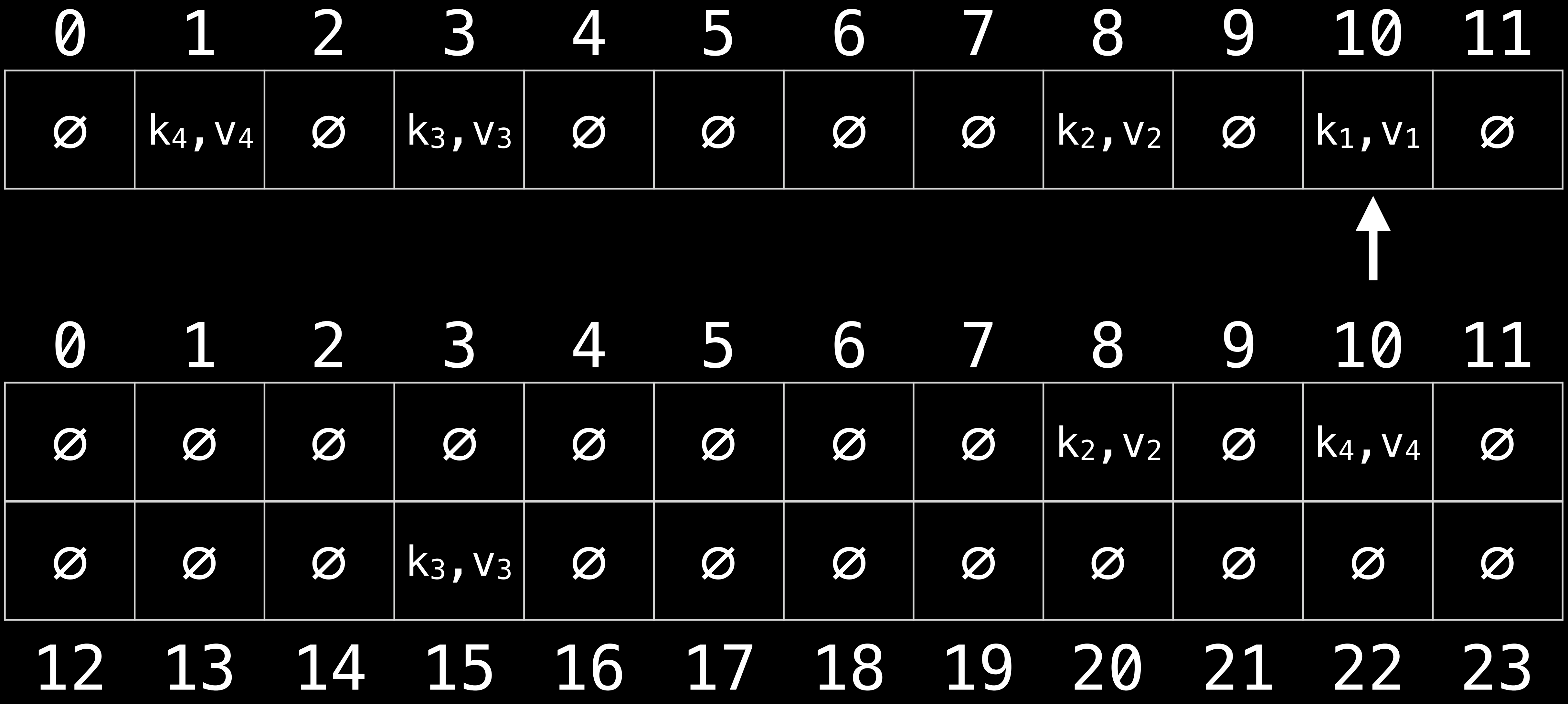
$\mathbf{H}(k_2) + \mathbf{P}(0) \bmod N = 8$

Recall $\mathbf{P}(x) = 5x$, $N = 24$, threshold $= 8$

0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅
									↑		

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	k_2,v_2	∅	k_4,v_4	∅
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

Recall **P**(x) = 5x, N = 24, threshold = 8



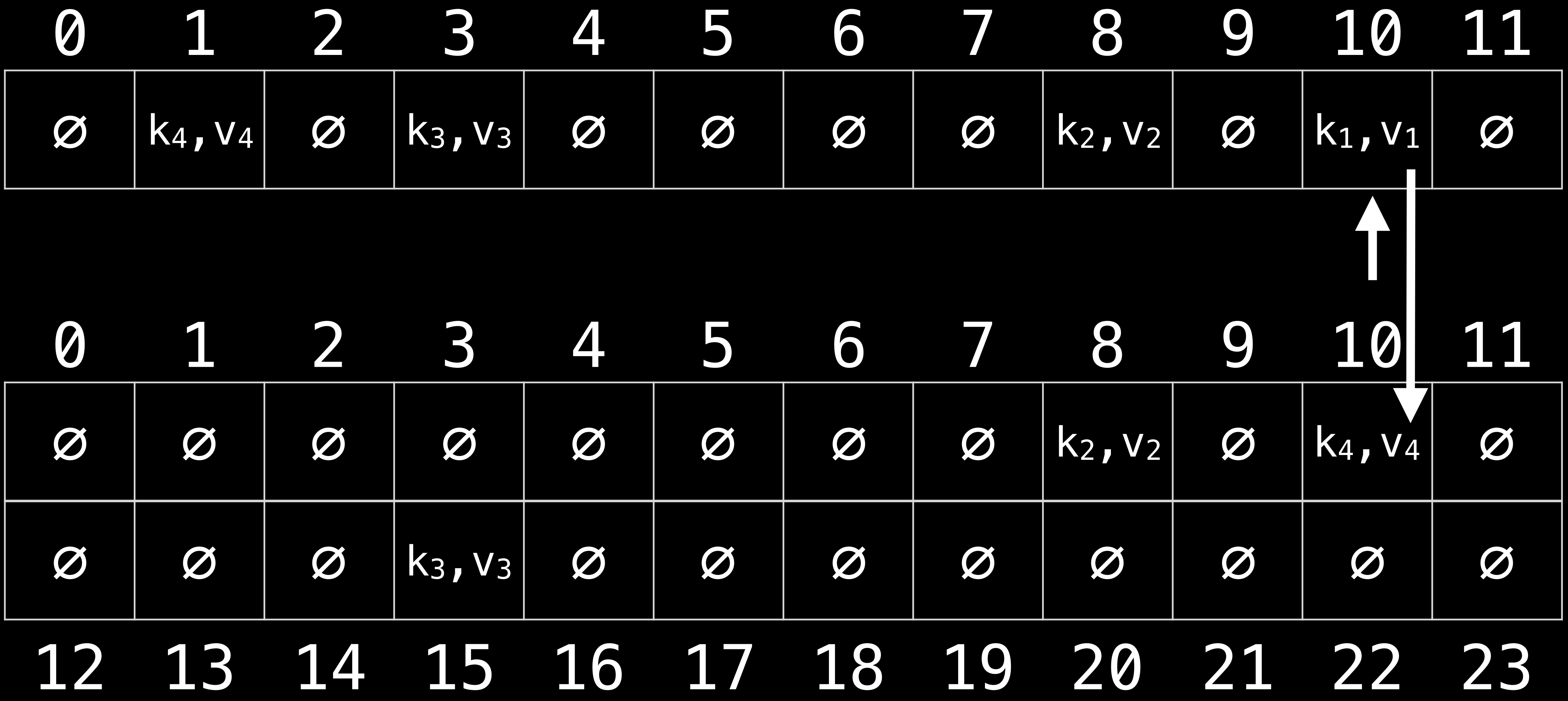
0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	k_2,v_2	∅	k_4,v_4	∅
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

From before, $\mathbf{H}(k_1) = 10$

$\mathbf{H}(k_1) + \mathbf{P}(0) \bmod N = 10$

Recall $\mathbf{P}(x) = 5x$, $N = 24$, threshold $= 8$



0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

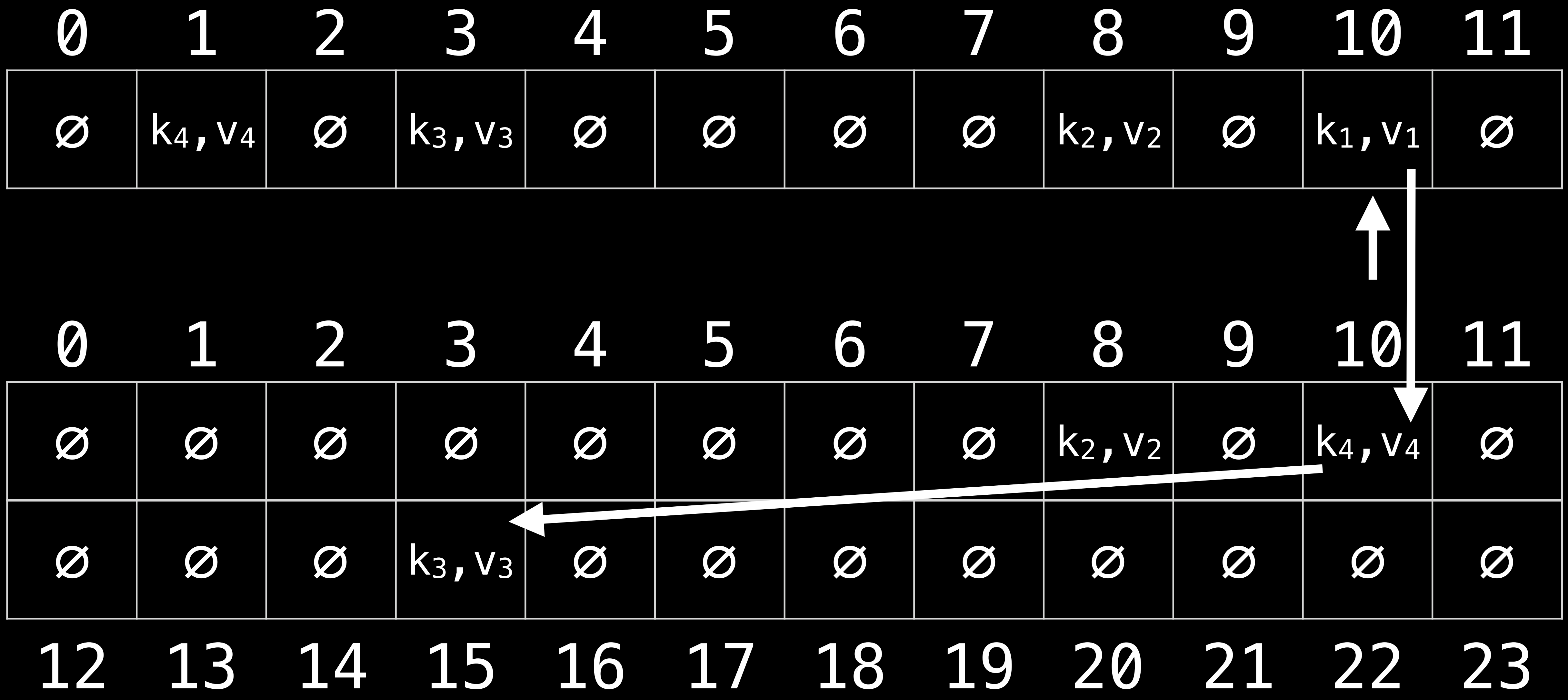
0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	k_2,v_2	∅	k_4,v_4	∅
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

From before, $H(k_1) = 10$

$H(k_1) + P(0) \bmod N = 10$

There’s a collision at position 10 in the new table, so keep probing.

Recall $P(x) = 5x$, $N = 24$, threshold = 8



0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

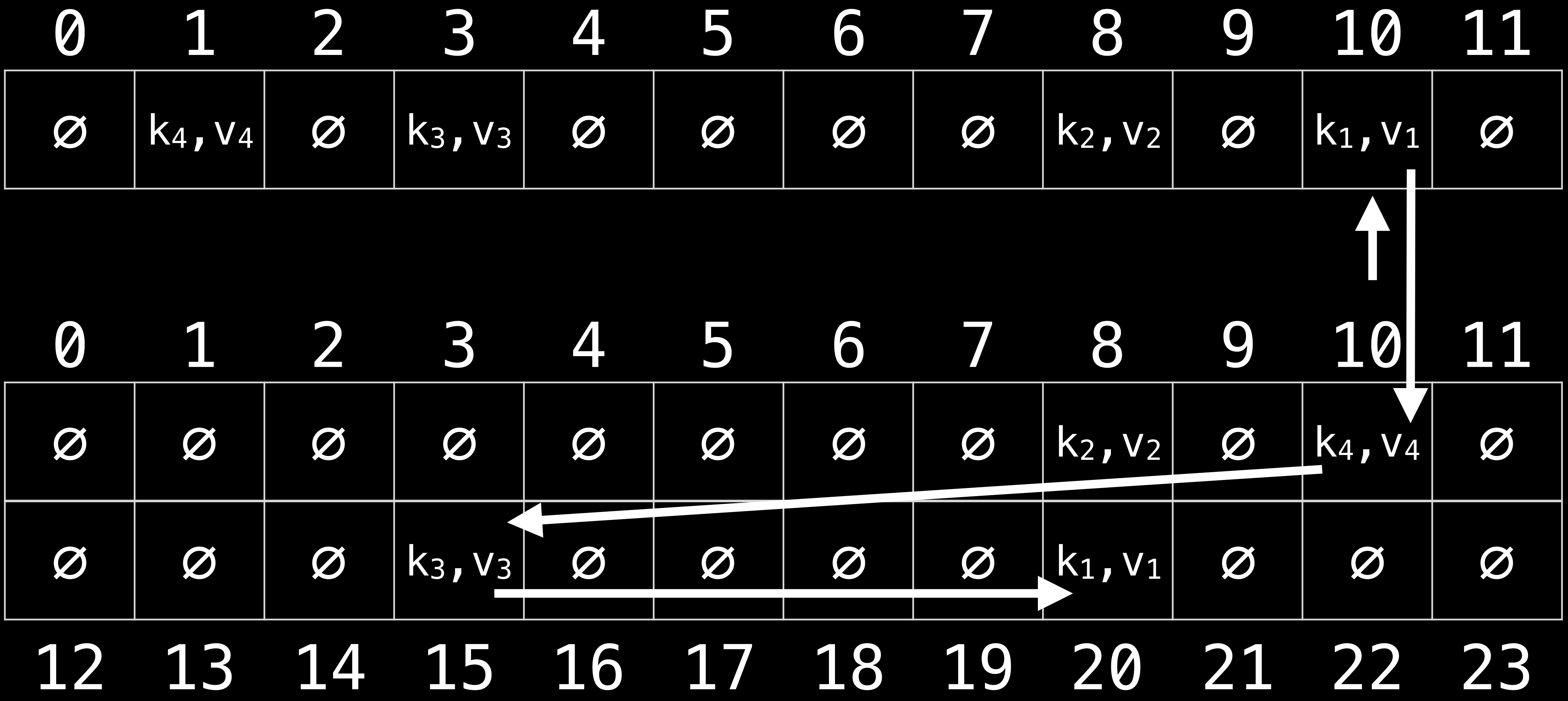
0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	k_2,v_2	∅	k_4,v_4	∅
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	∅	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

From before, $H(k_1) = 10$

$H(k_1) + P(0) \bmod N = 10$
$H(k_1) + P(1) \bmod N = 15$

There's a collision at position 15 in the new table, so keep probing.

Recall $P(x) = 5x$, $N = 24$, threshold = 8



0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4, v_4	∅	k_3, v_3	∅	∅	∅	∅	k_2, v_2	∅	k_1, v_1	∅

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	k_2, v_2	∅	k_4, v_4	∅
∅	∅	∅	k_3, v_3	∅	∅	∅	∅	k_1, v_1	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

From before, $\mathbf{H}(k_1) = 10$

$\mathbf{H}(k_1) + \mathbf{P}(0) \bmod N = 10$
$\mathbf{H}(k_1) + \mathbf{P}(1) \bmod N = 15$
$\mathbf{H}(k_1) + \mathbf{P}(2) \bmod N = 20$

Recall $\mathbf{P}(x) = 5x$, $N = 24$, threshold = 8

0	1	2	3	4	5	6	7	8	9	10	11
∅	k_4,v_4	∅	k_3,v_3	∅	∅	∅	∅	k_2,v_2	∅	k_1,v_1	∅

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	k_2,v_2	∅	k_4,v_4	∅
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	k_1,v_1	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

Recall P(x) = 5x, N = 24, threshold = 8

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	k_2,v_2	∅	k_4,v_4	∅
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	k_1,v_1	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

Recall P(x) = 5x, N = 24, threshold = 8

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_5,v_5)

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	∅	∅	∅	∅	∅	∅	k_2, v_2	∅	k_4, v_4	∅
∅	∅	∅	k_3, v_3	∅	∅	∅	∅	k_1, v_1	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

Recall $P(x) = 5x$, $N = 24$, threshold = 8

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_5,v_5)←

Suppose $H(k_5) = 2$

$H(k_5) + P(0) \bmod N = 2$

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	k_5,v_5	∅	∅	∅	∅	∅	k_2,v_2	∅	k_4,v_4	∅
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	k_1,v_1	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

Recall $P(x) = 5x$, $N = 24$, threshold = 8

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_5,v_5) ←

Suppose $H(k_5) = 2$

$H(k_5) + P(0) \bmod N = 2$

0	1	2	3	4	5	6	7	8	9	10	11
∅	∅	k_5,v_5	∅	∅	∅	∅	∅	k_2,v_2	∅	k_4,v_4	∅
∅	∅	∅	k_3,v_3	∅	∅	∅	∅	k_1,v_1	∅	∅	∅
12	13	14	15	16	17	18	19	20	21	22	23

Recall $P(x) = 5x$, $N = 24$, threshold = 8

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_5,v_5)

Suppose $H(k_5) = 2$

$H(k_5) + P(0) \bmod N = 2$

FAQ

Q: Sweet, I know how insertion works, now how do I removed key-value pairs from the hash table using open addressing?

FAQ

Q: Sweet, I know how insertion works, now how do I removed key–value pairs from the hash table using open addressing?

A: This topic by itself merits its own video (link in the description).

Next Video:
Open addressing quadratic probing

Multiple hash table implementations and source code and tests can all be found at:

github.com/williamfiset/data-structures

Hash table Quadratic Probing

An in depth look at quadratic probing

William Fiset

Open addressing main idea

General insertion method for open addressing on a table of size N goes as follows:

```
x := 1
keyHash := H(k) mod N
index := keyHash

while table[index] != null:
    index = (keyHash + P(k,x)) mod N
    x = x + 1

insert (k,v) at table[index]
```

Where H(k) is the hash for the key k and P(k,x) is the probing function

What is Quadratic Probing (QP)?

QP is a **probing method** which probes according to a quadratic formula, specifically:

P(x) = $ax^2 + bx + c$ where a,b,c are constants and $a \neq 0$ (otherwise we have linear probing)

(Note: The constant c is obsolete, do you know why?)

However, as we previously saw not all quadratic functions are viable because they are unable to produce a cycle of order **N**. We will need some way to handle this.

Chaos with cycles

Randomly selected QP functions have the issue that they easily produce short cycles. For example, if **P**(x) = 2x² + 2, **H**(k) = 4, and table size is nine (N = 9) we end up with the following cycle occurring:

H(k)+**P**(0) mod N = 4
H(k)+**P**(1) mod N = 7
H(k)+**P**(2) mod N = 4
H(k)+**P**(3) mod N = 7
H(k)+**P**(4) mod N = 4
H(k)+**P**(5) mod N = 7
H(k)+**P**(6) mod N = 4
H(k)+**P**(7) mod N = 7
H(k)+**P**(8) mod N = 4
…

The cycle {4,7} makes it impossible to reach buckets {0,1,2,3,5,6,8}!

This would cause an **infinite loop** in our hash table if the buckets 4 and 7 were already occupied!

Chaos with cycles

Q: So how do we pick a probing function we can work with?

A: There are numerous ways, but three of the most popular approaches are:

1) Let **P**$(x) = x^2$, keep the table size a prime number > 3 and also keep $\alpha \leq ½$

2) Let **P**$(x) = (x^2 + x)/2$ and keep the table size a power of two

3) Let **P**$(x) = (-1^x)*x^2$ and keep the table size a prime N where $N \equiv 3 \mod 4$

Chaos with cycles

Let's see an example of inserting using this quadratic probing function…

2) Let **P**(x) = $(x^2 + x)/2$ and keep the table size a power of two

Inserting with QP

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	∅	∅

Suppose we have an originally empty hash table and we want to insert some (k_i,v_i) pairs with QP and we selected our hash table to have:

P$(x) = (x^2 + x)/2$

Probing function: **P**$(x) = (x^2 + x)/2$
Table size: $N = 2^3 = 8$ (power of two)
Max load factor: $\alpha = 0.4$
Threshold before resize $= N * \alpha = 3$

Inserting with QP

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	∅	∅

Suppose we have an originally empty hash table and we want to insert some (k_i,v_i) pairs with QP and we selected our hash table to have:

Probing function: **P**$(x) = (x^2 + x)/2$
Table size: **$N = 2^3 = 8$ (power of two)**
Max load factor: $\alpha = 0.4$
Threshold before resize $= N * \alpha = 3$

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	∅	∅

Recall **P**(x) = $(x^2 + x)/2$, N = 8, threshold = 3

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	∅	∅

Recall **P**(x) = $(x^2 + x)/2$, N = 8, threshold = 3

Operations:

Suppose **H**(k_1) = 6

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	∅	∅

Recall $P(x) = (x^2 + x)/2$, $N = 8$, threshold = 3

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H(k_1) = 6$

$$H(k_1) + P(0) \bmod N$$

$$6 + 0 \bmod 8 = 6$$

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	k_1, v_1	∅
						↑	

Recall $P(x) = (x^2 + x)/2$, $N = 8$, threshold = 3

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H(k_1) = 6$

$H(k_1) + P(0) \bmod N$

$6 + 0 \bmod 8 = 6$

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	k_1, v_1	∅

Recall **P**$(x) = (x^2 + x)/2$, $N = 8$, threshold = 3

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	k_1,v_1	∅

Recall $P(x) = (x^2 + x)/2$, $N = 8$, threshold = 3

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H(k_2) = 5$

$$H(k_2) + P(0) \bmod N$$

$$5 + 0 \bmod 8 = 5$$

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_2, v_2	k_1, v_1	∅
					↑		

Recall $P(x) = (x^2 + x)/2$, $N = 8$, threshold = 3

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H(k_2) = 5$

$H(k_2) + P(0) \bmod N$

$5 + 0 \bmod 8 = 5$

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

Recall $P(x) = (x^2 + x)/2$, $N = 8$, threshold = 3

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

Recall $P(x) = (x^2 + x)/2$, $N = 8$, threshold = 3

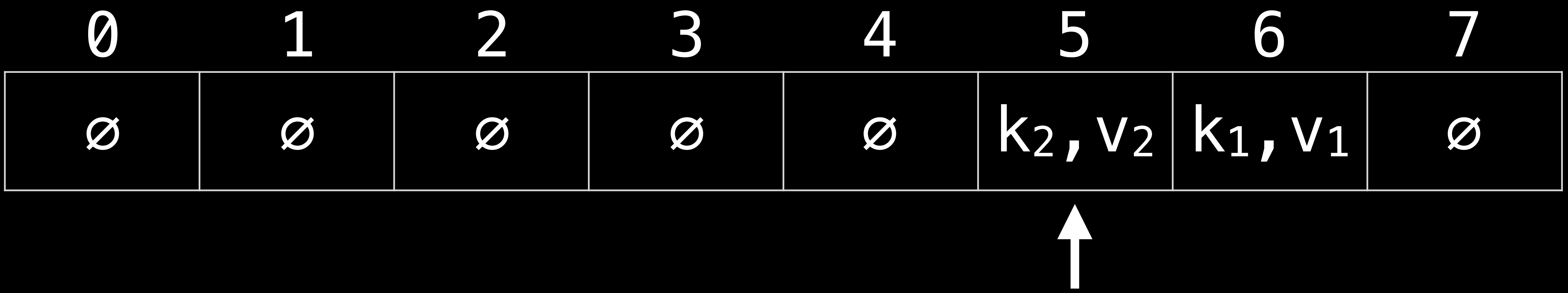
Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H(k_3) = 5$

$H(k_3) + P(0) \bmod N$

$5 + 0 \bmod 8 = 5$



0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

Recall $P(x) = (x^2 + x)/2$, $N = 8$, threshold = 3

<u>Operations</u>:

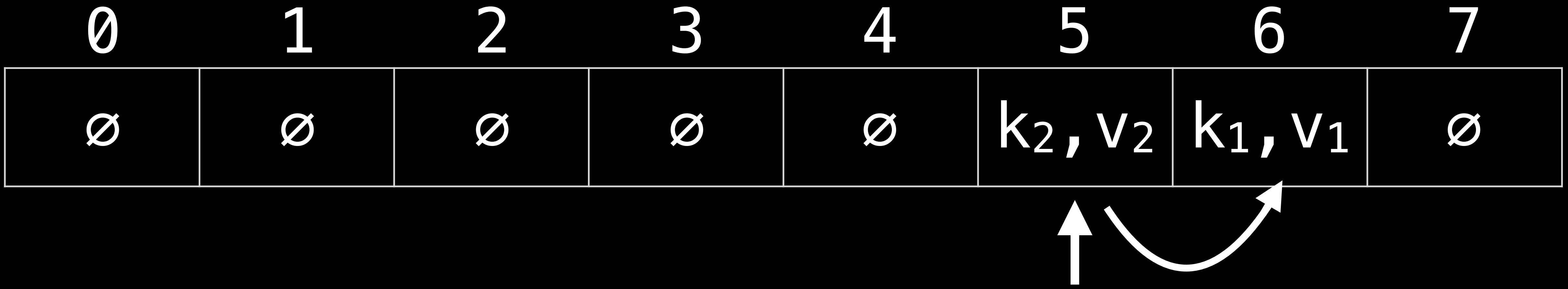
insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H(k_3) = 5$

$H(k_3) + P(0) \bmod N$

$5 + 0 \bmod 8 = 5$

Bucket 5 is already taken! Try the next probe position at $P(1)$ instead of $P(0)$



0	1	2	3	4	5	6	7
⌀	⌀	⌀	⌀	⌀	k_2,v_2	k_1,v_1	⌀

Recall $P(x) = (x^2 + x)/2$, N = 8, threshold = 3

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H(k_3) = 5$

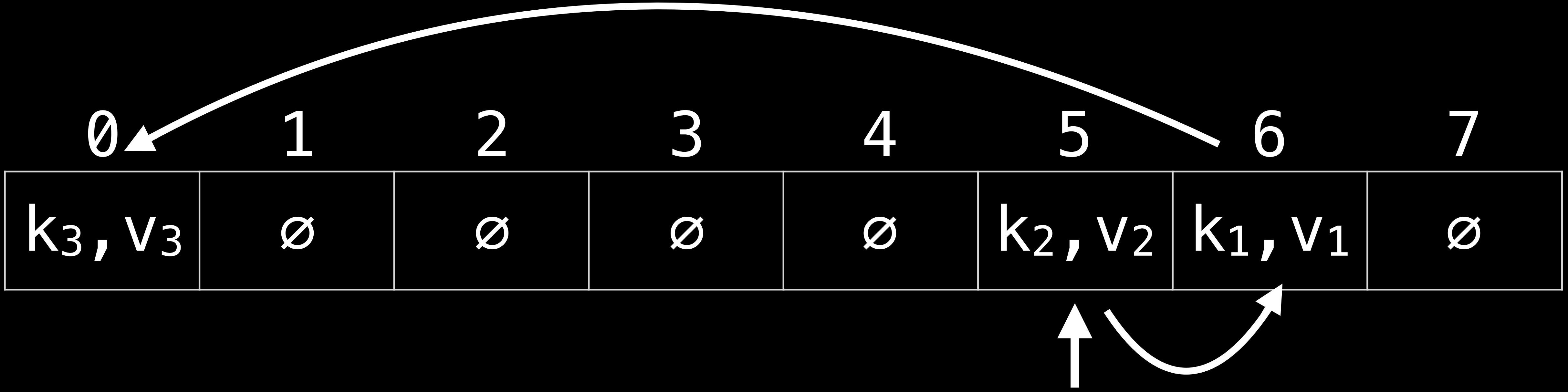
$H(k_3) + P(0) \bmod N$

$5 + 0 \bmod 8 = 5$

$H(k_3) + P(1) \bmod N$

$5 + 1 \bmod 8 = 6$

Bucket 6 is already taken! Try the next probe position at $P(2)$ instead of $P(1)$



0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

Recall $P(x) = (x^2 + x)/2$, $N = 8$, threshold $= 3$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H(k_3) = 5$

$H(k_3) + P(0) \bmod N$

$5 + 0 \bmod 8 = 5$

$H(k_3) + P(1) \bmod N$

$5 + 1 \bmod 8 = 6$

$H(k_3) + P(2) \bmod N$

$5 + 3 \bmod 8 = 0$

0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

Recall **P**$(x) = (x^2 + x)/2$, $N = 8$, threshold $= 3$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

Recall **P**$(x) = (x^2 + x)/2$, $N = 8$, **threshold = 3**

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

We have now reached the table **threshold**, so it’s time to resize the table!

0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	∅	∅
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

Our quadratic probing scheme requires that the table size remains a power of two, so let’s double the table size!

0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	∅	∅
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

New table size $N = 2^4 = 16$

Max load factor $\alpha = 0.4$

New threshold value $= N*\alpha = 6$

Probing function **P**(x) stays the same

0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

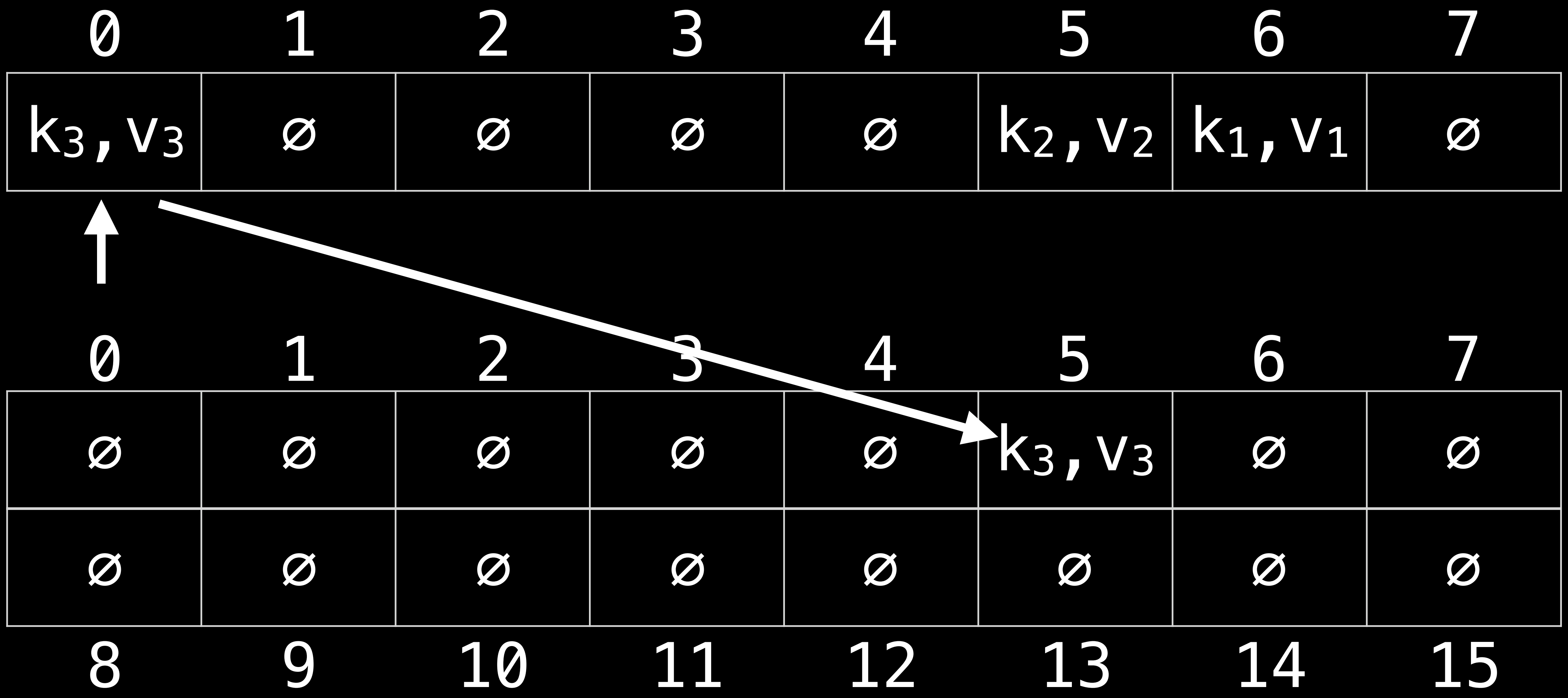
0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	∅	∅
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

↑

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	∅	∅
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

From before, **H**(k_3) = 5



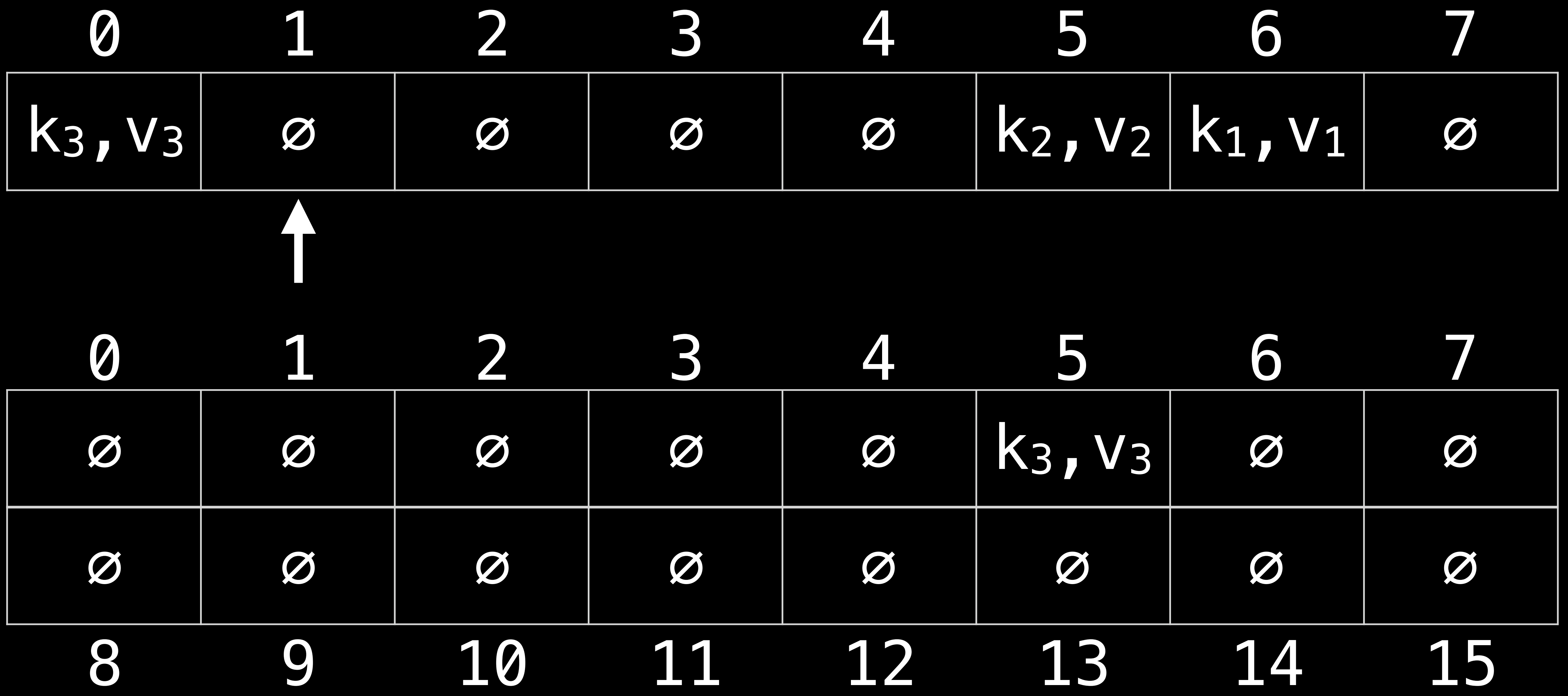
0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_3,v_3	∅	∅
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

From before, $\mathbf{H}(k_3) = 5$

$\mathbf{H}(k_3) + \mathbf{P}(0) \bmod N$

$5 + 0 \bmod 16 = 5$



0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_3,v_3	∅	∅
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

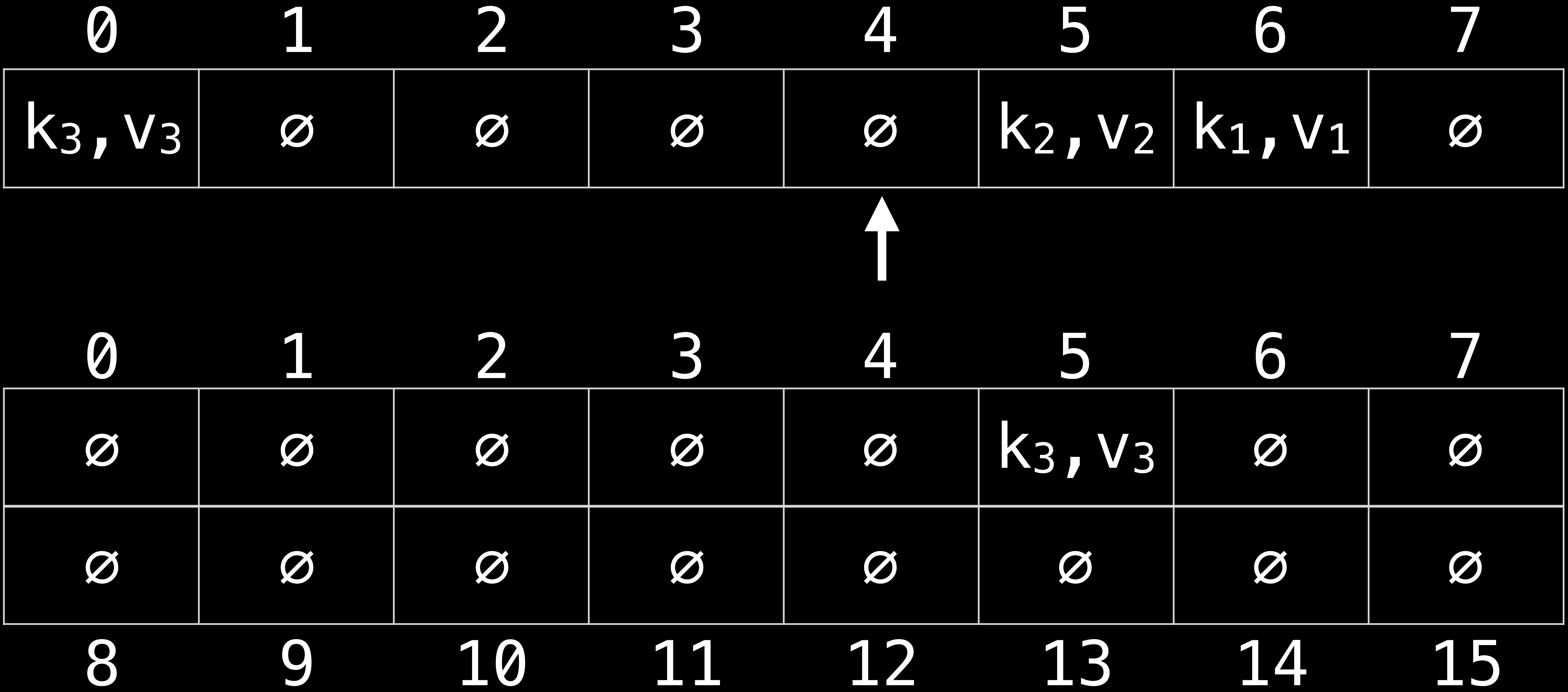
0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅
		↑					

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_3,v_3	∅	∅
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

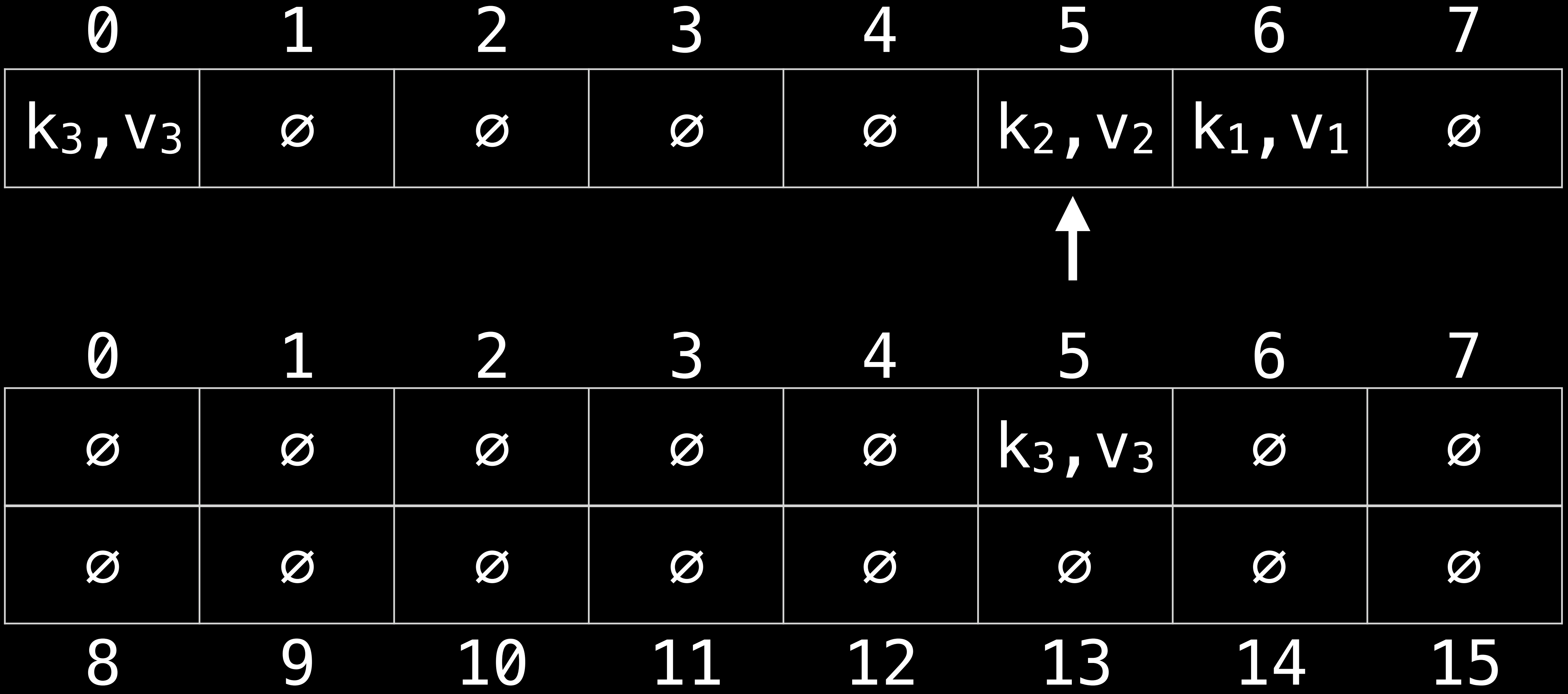
↑

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_3,v_3	∅	∅
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15



0	1	2	3	4	5	6	7
k_3, v_3	∅	∅	∅	∅	k_2, v_2	k_1, v_1	∅

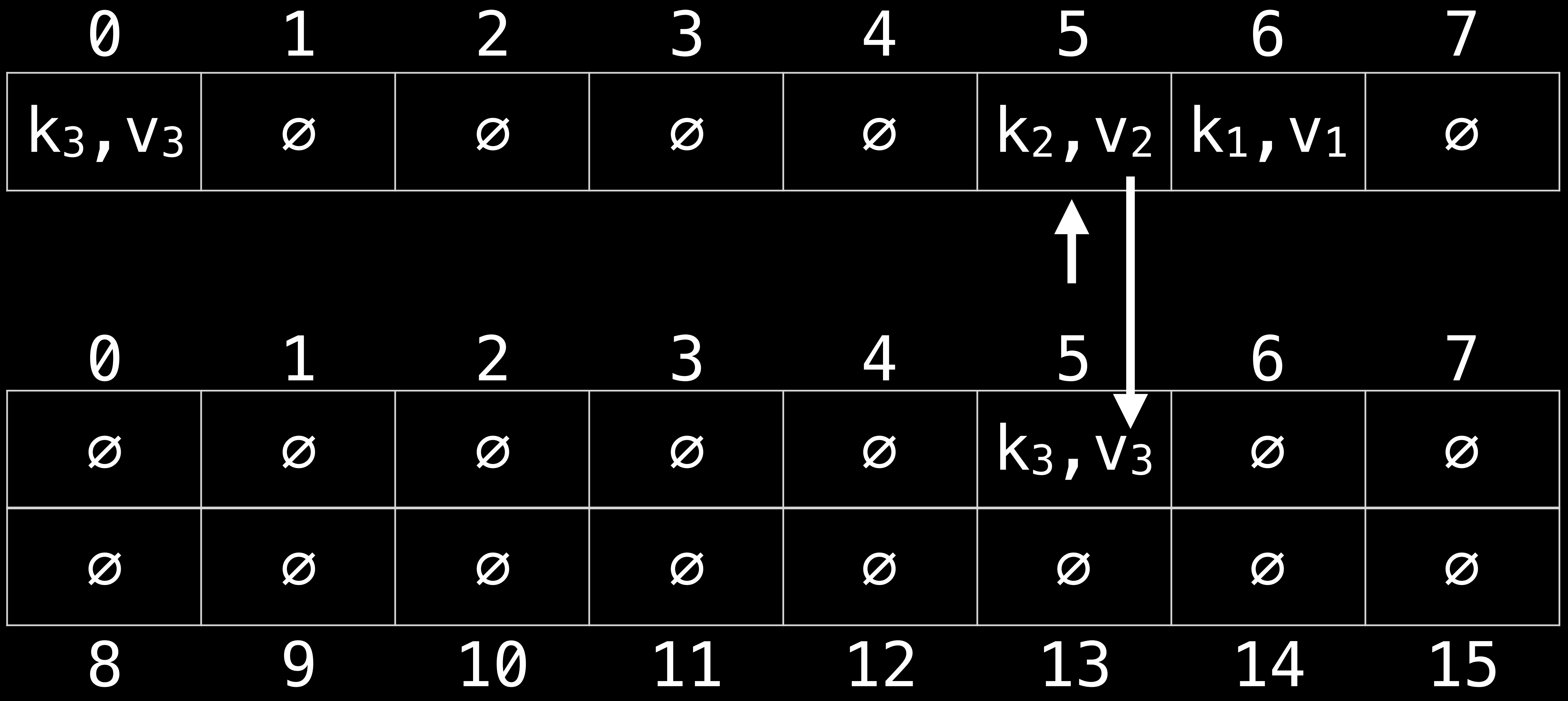
0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_3, v_3	∅	∅
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15



0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

↑

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_3,v_3	∅	∅
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15



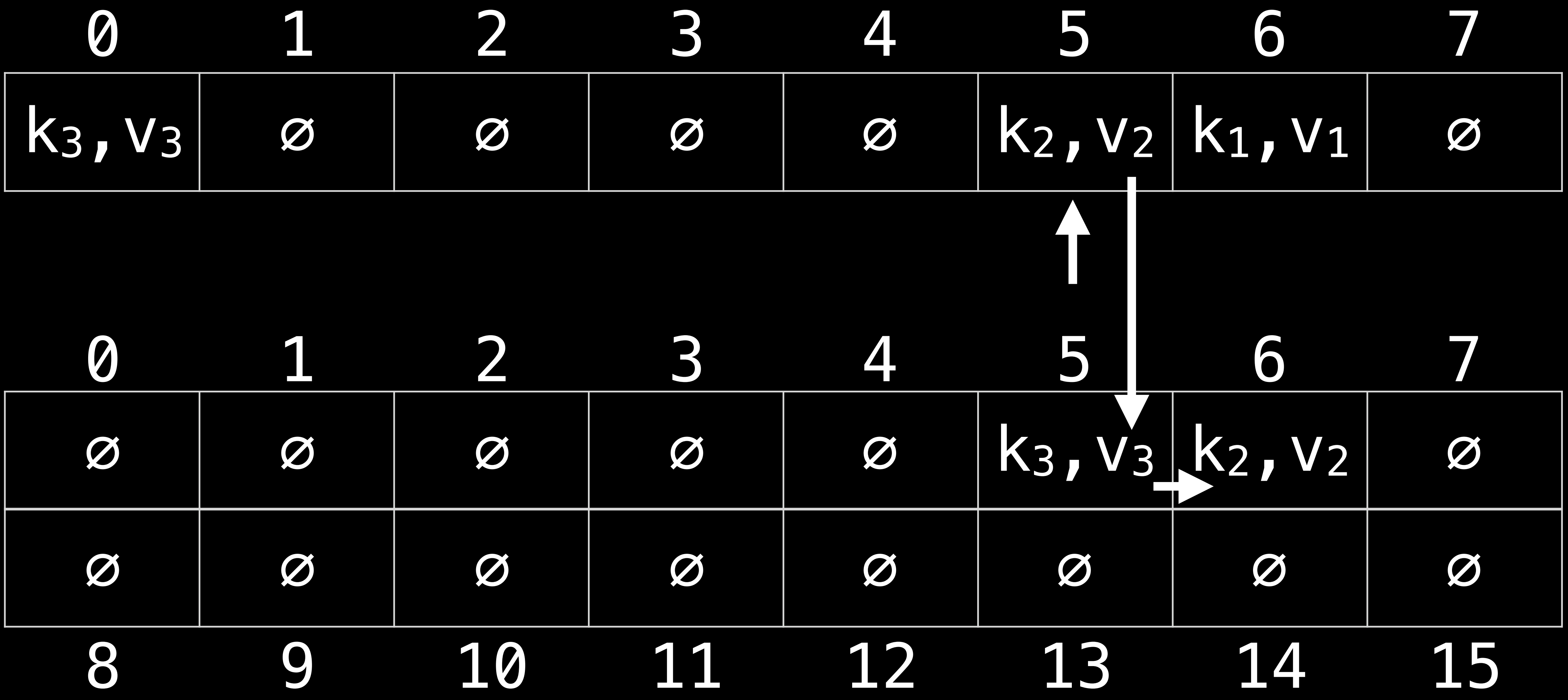
0	1	2	3	4	5	6	7
k_3, v_3	∅	∅	∅	∅	k_2, v_2	k_1, v_1	∅

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_3, v_3	∅	∅
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

From before, $\mathbf{H}(k_2) = 5$

$\mathbf{H}(k_2) + \mathbf{P}(0) \bmod N$

$5 + 0 \bmod 16 = 5$



0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_3,v_3	k_2,v_2	∅
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

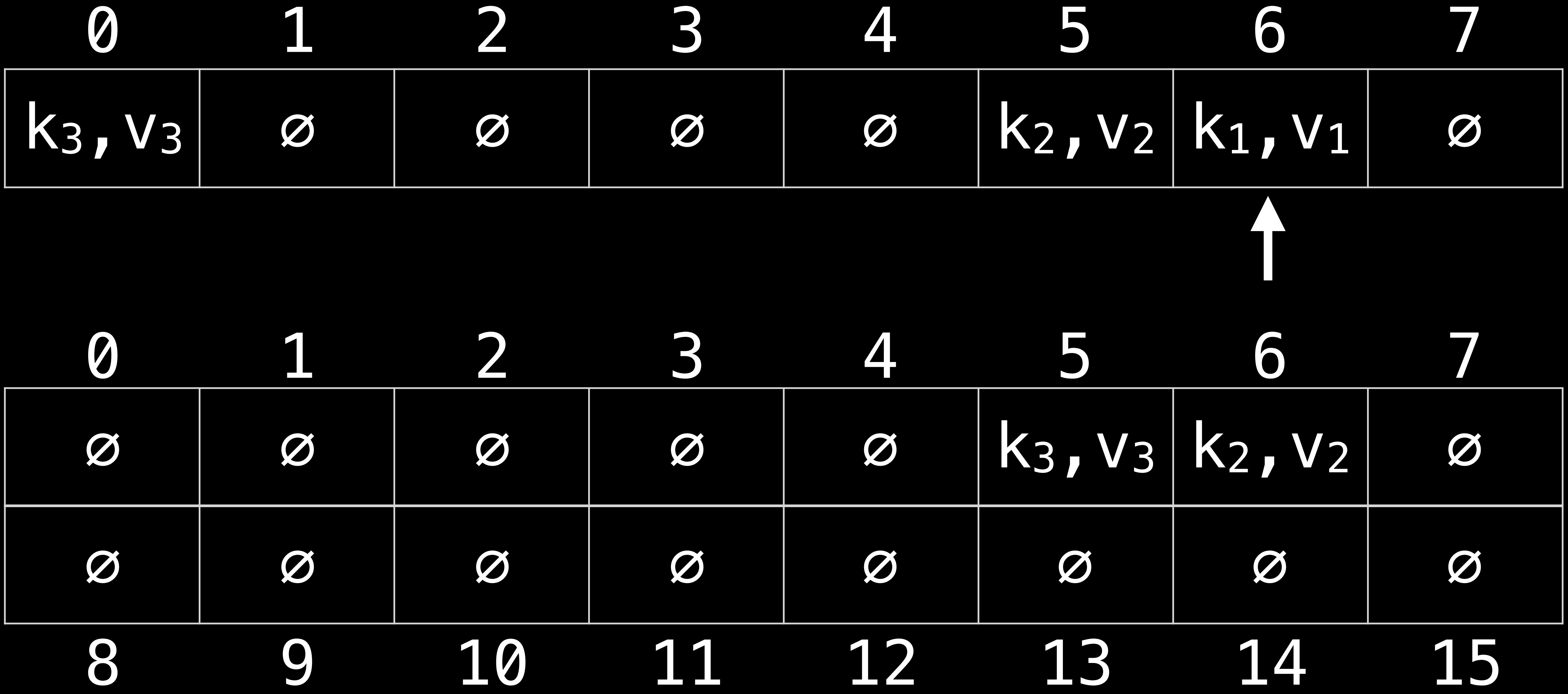
From before, $\mathbf{H}(k_2) = 5$

$\mathbf{H}(k_2) + \mathbf{P}(0) \bmod N$

$5 + 0 \bmod 16 = 5$

$\mathbf{H}(k_2) + \mathbf{P}(1) \bmod N$

$5 + 1 \bmod 16 = 6$



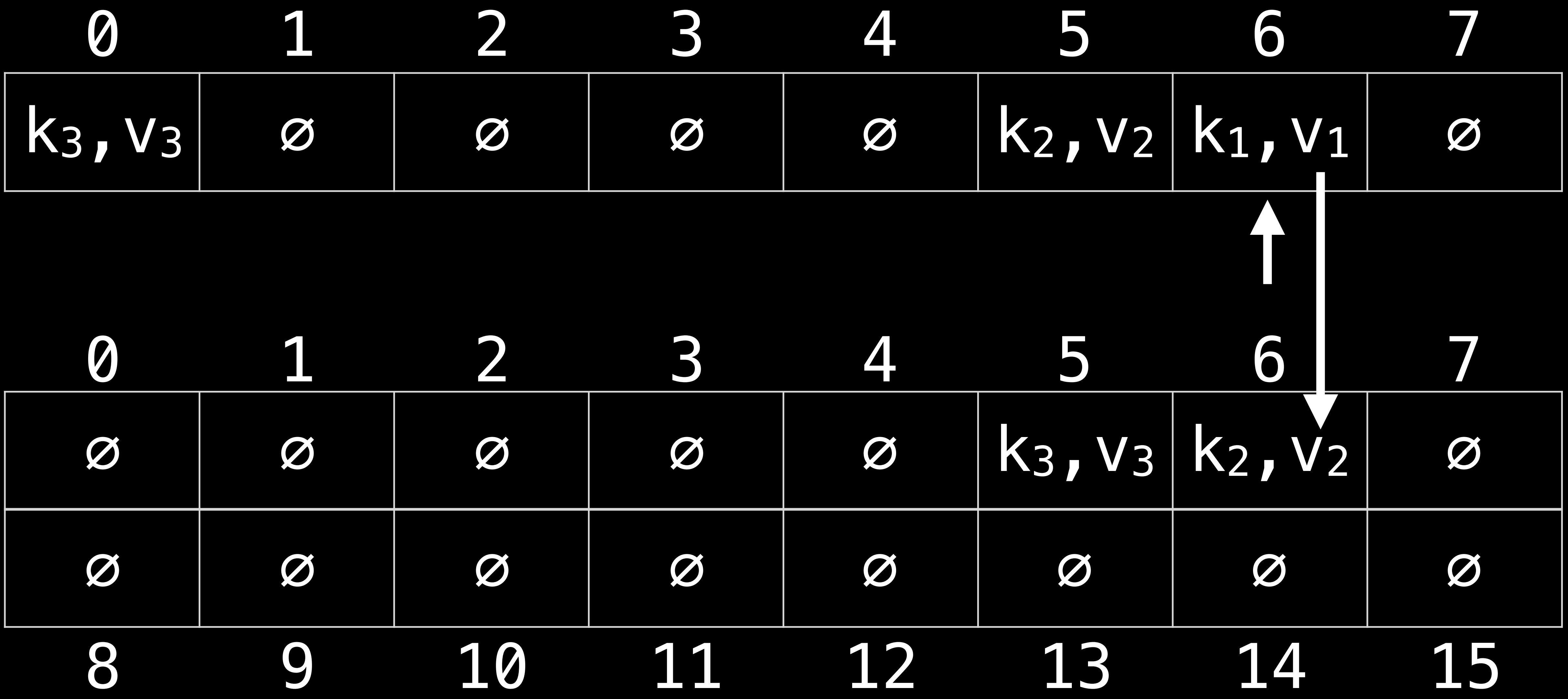
0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_3,v_3	k_2,v_2	∅
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅
						↑	

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_3,v_3	k_2,v_2	∅
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

From before, **H**(k_1) = 6



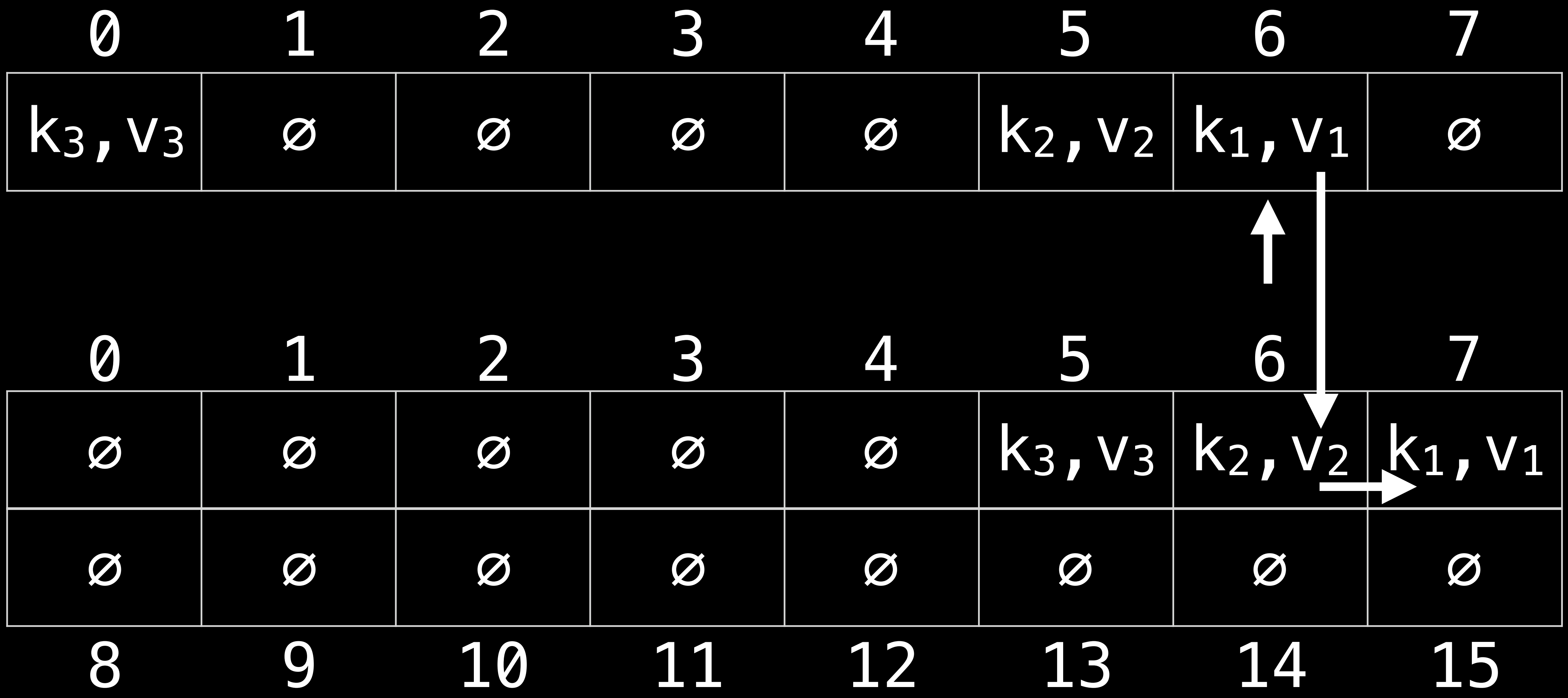
0	1	2	3	4	5	6	7
k_3, v_3	∅	∅	∅	∅	k_2, v_2	k_1, v_1	∅

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_3, v_3	k_2, v_2	∅
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

From before, $H(k_1) = 6$

$H(k_1) + P(0) \bmod N$

$6 + 0 \bmod 16 = 6$



0	1	2	3	4	5	6	7
k_3, v_3	∅	∅	∅	∅	k_2, v_2	k_1, v_1	∅

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_3, v_3	k_2, v_2	k_1, v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

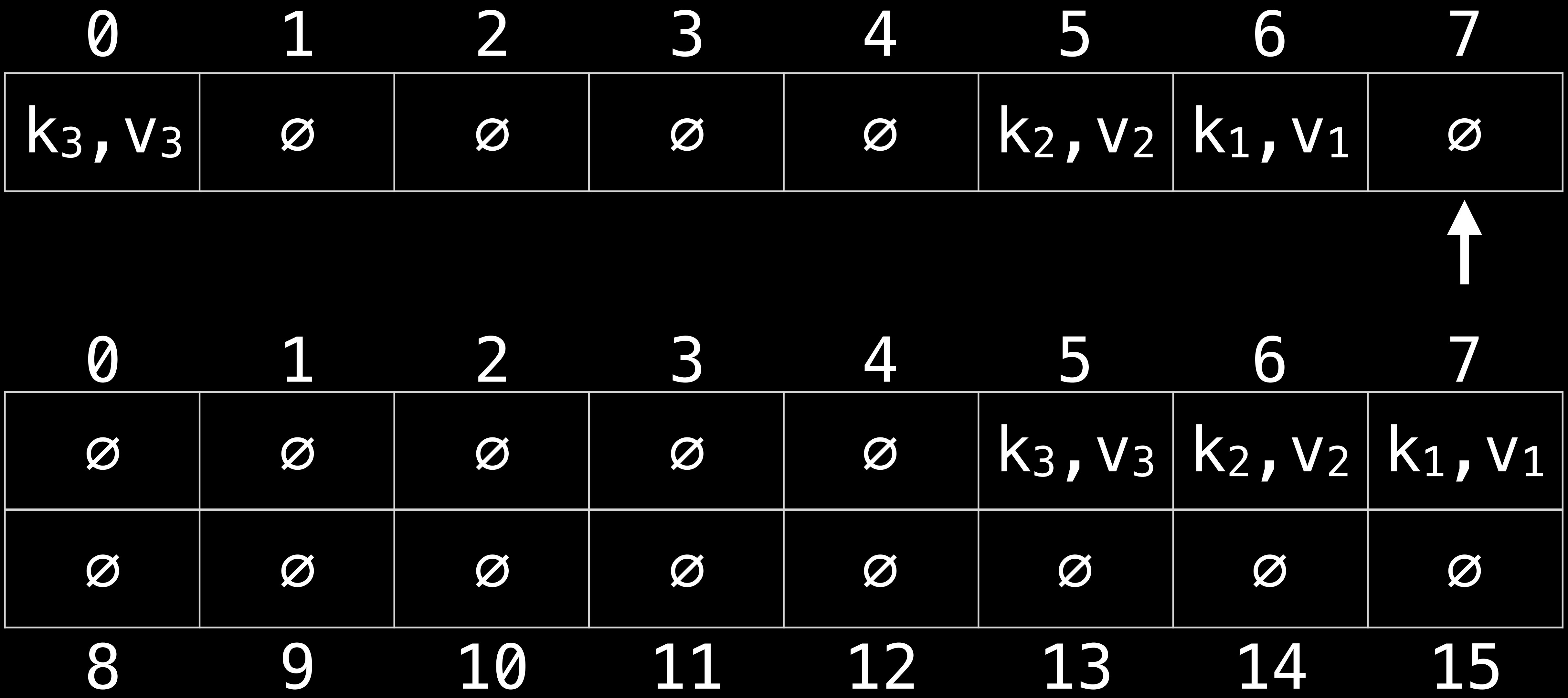
From before, $H(k_1) = 6$

$H(k_1) + P(0) \bmod N$

$6 + 0 \bmod 16 = 6$

$H(k_1) + P(1) \bmod N$

$6 + 1 \bmod 16 = 7$



0	1	2	3	4	5	6	7
k_3,v_3	∅	∅	∅	∅	k_2,v_2	k_1,v_1	∅

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_3,v_3	k_2,v_2	k_1,v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_3,v_3	k_2,v_2	k_1,v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

Recall $P(x) = (x^2 + x)/2$, $N = 16$, threshold = 6

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	k_3,v_3	k_2,v_2	k_1,v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

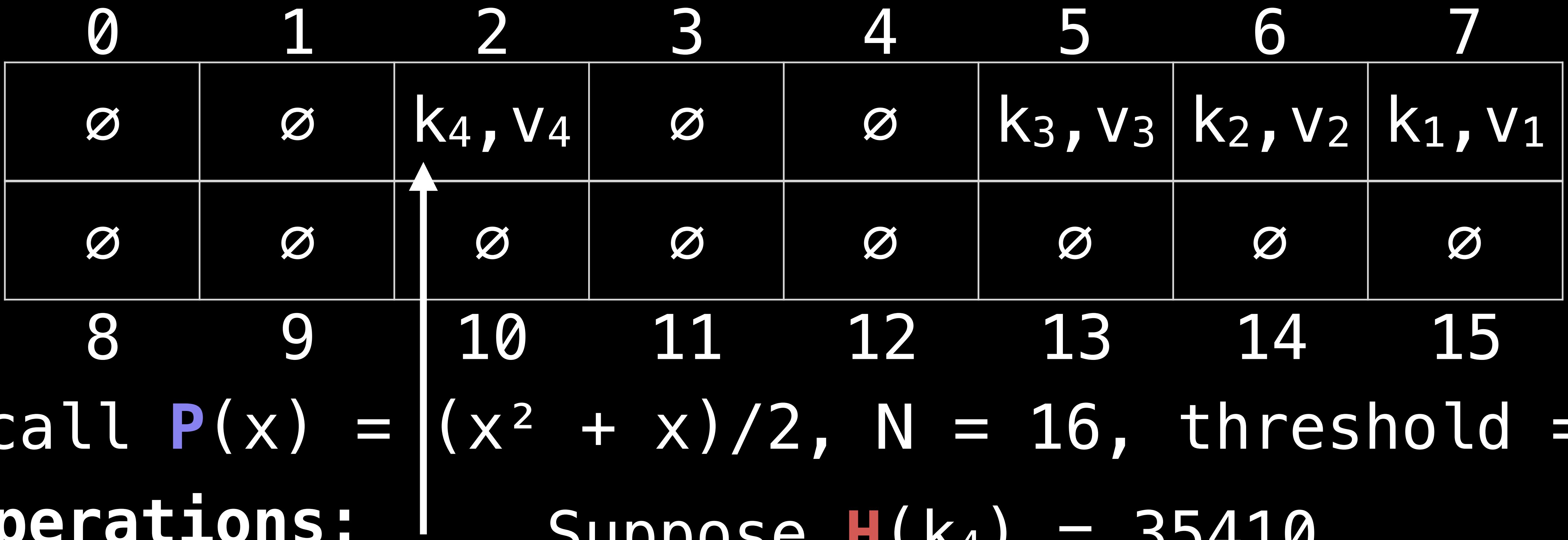
Recall $\mathbf{P}(x) = (x^2 + x)/2$, $N = 16$, threshold = 6

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $\mathbf{H}(k_4) = 35410$

$\mathbf{H}(k_4) + \mathbf{P}(0) \bmod N =$
$35410 + 0 \bmod 16 = 2$



0	1	2	3	4	5	6	7
∅	∅	k_4, v_4	∅	∅	k_3, v_3	k_2, v_2	k_1, v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

Recall $P(x) = (x^2 + x)/2$, $N = 16$, threshold = 6

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H(k_4) = 35410$

$H(k_4) + P(0) \bmod N =$
$35410 + 0 \bmod 16 = 2$

0	1	2	3	4	5	6	7
∅	∅	k_4, v_4	∅	∅	k_3, v_3	k_2, v_2	k_1, v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

Recall $P(x) = (x^2 + x)/2$, $N = 16$, threshold = 6

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6	7
∅	∅	k_4, v_4	∅	∅	k_3, v_3	k_2, v_2	k_1, v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

Recall $P(x) = (x^2 + x)/2$, $N = 16$, threshold = 6

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

From before, $H(k_3) = 5$

0	1	2	3	4	5	6	7
∅	∅	k_4,v_4	∅	∅	$k_3,$**v_5**	k_2,v_2	k_1,v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

Recall **P**$(x) = (x^2 + x)/2$, N = 16, threshold = 6

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

From before, **H**(k_3) = 5

H(k_3) + **P**(0) mod N
5 + 0 mod 16 = 5

Since k_3 already existed in the table simply update the value associated with k_3.

0	1	2	3	4	5	6	7
∅	∅	k_4,v_4	∅	∅	k_3,v_5	k_2,v_2	k_1,v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

Recall $P(x) = (x^2 + x)/2$, $N = 16$, threshold = 6

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6	7
∅	∅	k_4,v_4	∅	∅	k_3,v_5	k_2,v_2	k_1,v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

Recall $P(x) = (x^2 + x)/2$, $N = 16$, threshold = 6

Operations:

$\text{insert}(k_1,v_1)$
$\text{insert}(k_2,v_2)$
$\text{insert}(k_3,v_3)$
$\text{insert}(k_4,v_4)$
$\text{insert}(k_3,v_5)$
$\text{insert}(k_6,v_6)$
$\text{insert}(k_7,v_7)$

Suppose $H(k_6) = -6413$

0	1	2	3	4	5	6	7
∅	∅	k_4, v_4	∅	∅	k_3, v_5	k_2, v_2	k_1, v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

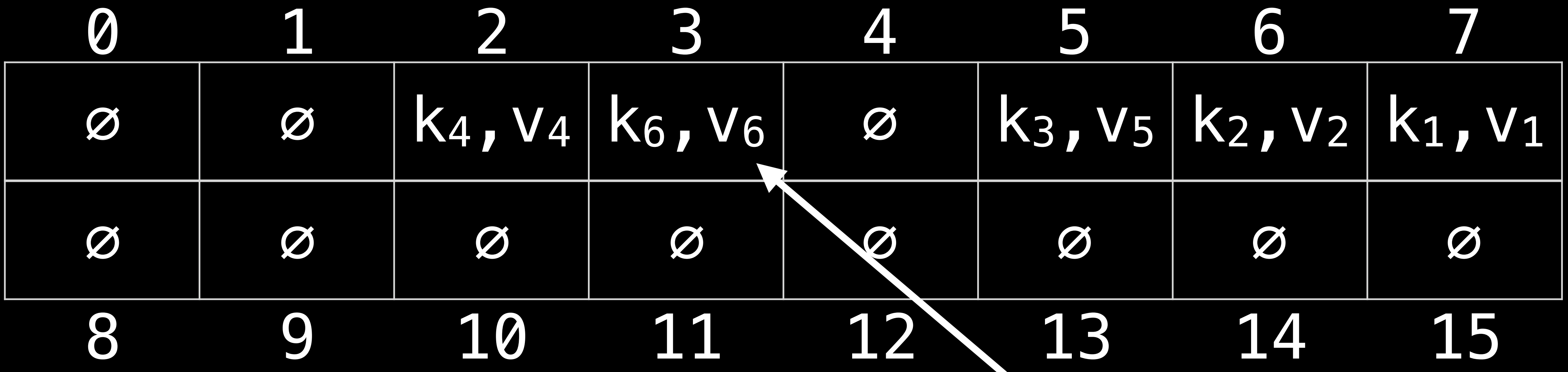
Recall $P(x) = (x^2 + x)/2$, $N = 16$, threshold $= 6$

Operations:

insert(k_1, v_1)
insert(k_2, v_2)
insert(k_3, v_3)
insert(k_4, v_4)
insert(k_3, v_5)
insert(k_6, v_6)
insert(k_7, v_7)

Suppose $H(k_6) = -6413$

$H(k_6) + P(0) \bmod N$
$-6413 + 0 \bmod 16 = 3$



0	1	2	3	4	5	6	7
∅	∅	k_4,v_4	k_6,v_6	∅	k_3,v_5	k_2,v_2	k_1,v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

Recall $P(x) = (x^2 + x)/2$, $N = 16$, threshold = 6

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H(k_6) = -6413$

$H(k_6) + P(0) \bmod N$
$-6413 + 0 \bmod 16 = 3$

0	1	2	3	4	5	6	7
∅	∅	k_4,v_4	k_6,v_6	∅	k_3,v_5	k_2,v_2	k_1,v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

Recall $P(x) = (x^2 + x)/2$, $N = 16$, threshold = 6

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6	7
∅	∅	k_4,v_4	k_6,v_6	∅	k_3,v_5	k_2,v_2	k_1,v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

Recall $P(x) = (x^2 + x)/2$, $N = 16$, threshold = 6

Operations:

Suppose $H(k_7) = 2$

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6	7
∅	∅	k_4,v_4	k_6,v_6	∅	k_3,v_5	k_2,v_2	k_1,v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

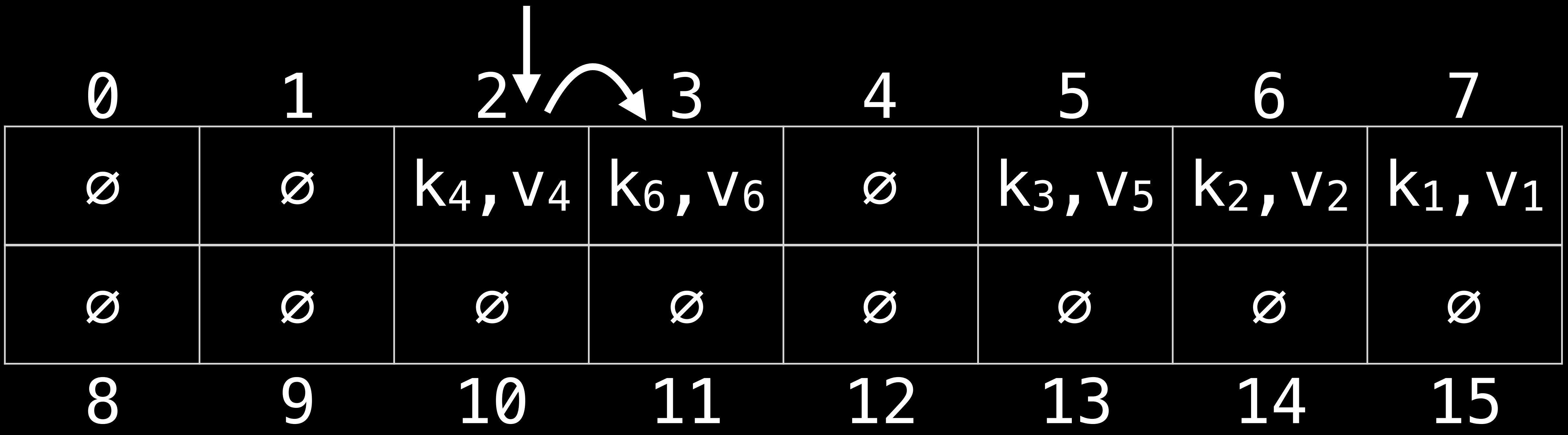
Recall $P(x) = (x^2 + x)/2$, $N = 16$, threshold = 6

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H(k_7) = 2$

$H(k_7) + P(0) \bmod N$
$2 + 0 \bmod 16 = 2$



0	1	2	3	4	5	6	7
∅	∅	k_4,v_4	k_6,v_6	∅	k_3,v_5	k_2,v_2	k_1,v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

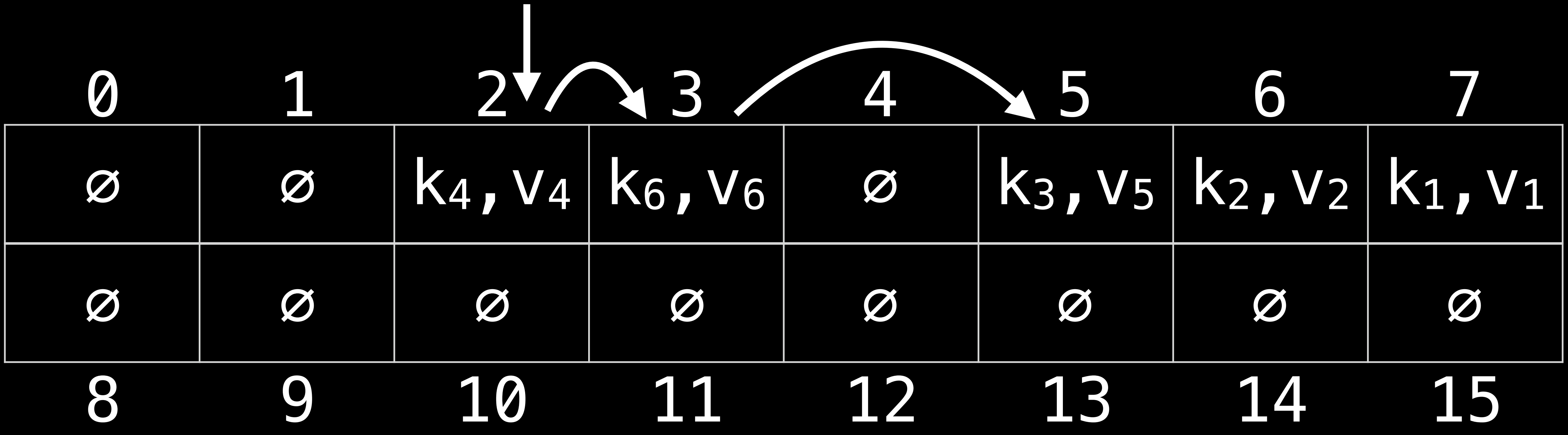
Recall $P(x) = (x^2 + x)/2$, $N = 16$, threshold = 6

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H(k_7) = 2$

$H(k_7) + P(0) \bmod N$
$2 + 0 \bmod 16 = 2$
$H(k_7) + P(1) \bmod N$
$2 + 1 \bmod 16 = 3$



0	1	2	3	4	5	6	7
∅	∅	k_4,v_4	k_6,v_6	∅	k_3,v_5	k_2,v_2	k_1,v_1
∅	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

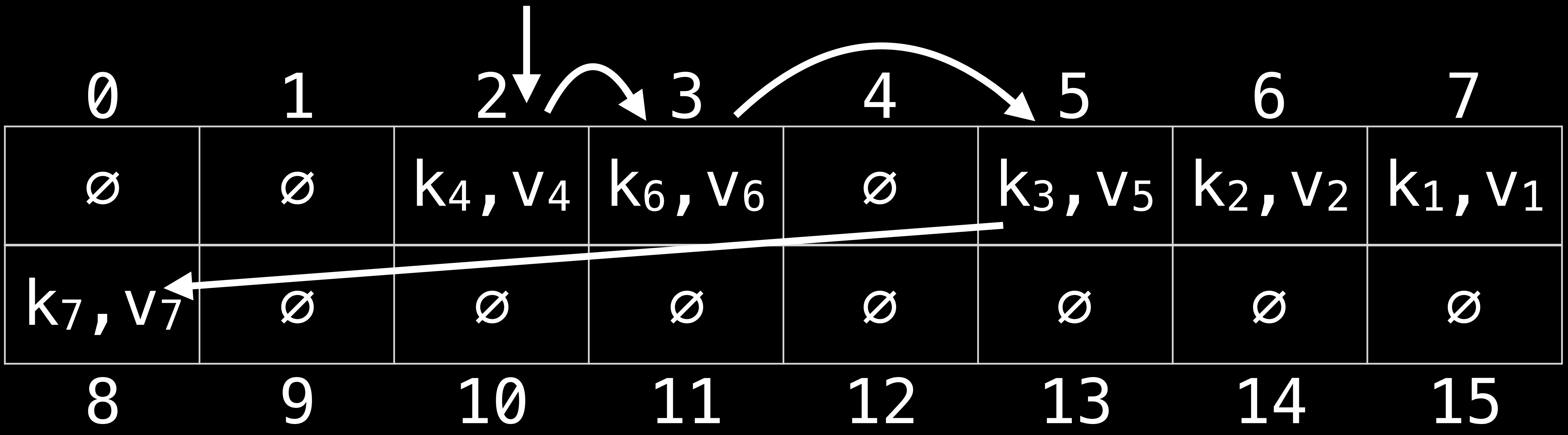
Recall $P(x) = (x^2 + x)/2$, $N = 16$, threshold = 6

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H(k_7) = 2$

$H(k_7) + P(0) \bmod N$
$2 + 0 \bmod 16 = 2$
$H(k_7) + P(1) \bmod N$
$2 + 1 \bmod 16 = 3$
$H(k_7) + P(2) \bmod N$
$2 + 3 \bmod 16 = 5$



0	1	2	3	4	5	6	7
∅	∅	k_4,v_4	k_6,v_6	∅	k_3,v_5	k_2,v_2	k_1,v_1
k_7,v_7	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

Recall $P(x) = (x^2 + x)/2$, N = 16, threshold = 6

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H(k_7) = 2$

$H(k_7) + P(0) \bmod N$
$2 + 0 \bmod 16 = 2$
$H(k_7) + P(1) \bmod N$
$2 + 1 \bmod 16 = 3$
$H(k_7) + P(2) \bmod N$
$2 + 3 \bmod 16 = 5$
$H(k_7) + P(3) \bmod N$
$2 + 6 \bmod 16 = 8$

0	1	2	3	4	5	6	7
∅	∅	k_4,v_4	k_6,v_6	∅	k_3,v_5	k_2,v_2	k_1,v_1
k_7,v_7	∅	∅	∅	∅	∅	∅	∅
8	9	10	11	12	13	14	15

Recall $P(x) = (x^2 + x)/2$, $N = 16$, threshold = 6

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Next Video:
Open addressing double hashing

Multiple hash table implementations and source code and tests can all be found at:

github.com/williamfiset/data-structures

Hash table (HT) Double Hashing

An in depth look at double hashing

William Fiset

Open addressing main idea

General insertion method for open addressing on a table of size **N** goes as follows:

```
x := 1
keyHash := $H_1$(k) mod N
index := keyHash

while table[index] != null:
    index = (keyHash + P(k,x)) mod N
    x = x + 1

insert (k,v) at table[index]
```

Where H_1(k) is the hash for the key k and **P**(k,x) is the probing function

What is Double Hashing (DH)?

DH is a **probing method** which probes according to a constant multiple of another hash function, specifically:

$P(k,x) = x*H_2(k)$, where $H_2(k)$ is a second hash function

What is Double Hashing (DH)?

DH is a **probing method** which probes according to a constant multiple of another hash function, specifically:

$P(k,x) = x*H_2(k)$, where $H_2(k)$ is a second hash function

$H_2(k)$ must hash the same type of keys as $H_1(k)$

NOTE: Notice that doubling hashing reduces to linear probing (except that the constant is unknown until runtime)

Chaos with cycles

Since DH reduces to linear probing at runtime we may end up with a linear probing function such as: $P(x) = 3x$, $H_1(k) = 4$, and table size is nine ($N = 9$) in which case we end up with the following cycle occurring:

$H(k)+P(0) \bmod N = 4$
$H(k)+P(1) \bmod N = 7$
$H(k)+P(2) \bmod N = 1$
$H(k)+P(3) \bmod N = 4$
$H(k)+P(4) \bmod N = 7$
$H(k)+P(5) \bmod N = 1$
$H(k)+P(6) \bmod N = 4$
$H(k)+P(7) \bmod N = 7$
$H(k)+P(8) \bmod N = 1$

…

Chaos with cycles

Since DH reduces to linear probing at runtime we may end up with a linear probing function such as: $P(x) = 3x$, $H_1(k) = 4$, and table size is nine $(N = 9)$ in which case we end up with the following cycle occurring:

$H(k)+P(0) \bmod N = 4$
$H(k)+P(1) \bmod N = 7$
$H(k)+P(2) \bmod N = 1$
$H(k)+P(3) \bmod N = 4$
$H(k)+P(4) \bmod N = 7$
$H(k)+P(5) \bmod N = 1$
$H(k)+P(6) \bmod N = 4$
$H(k)+P(7) \bmod N = 7$
$H(k)+P(8) \bmod N = 1$
…

The cycle {4,7,1} makes it impossible to reach buckets {0,2,3,5,6,8}!

This would cause an **infinite loop** in our hash table if all the buckets 4, 7, and 1 were already occupied!

Chaos with cycles

To fix the issue of cycles pick the table size to be a prime number and also compute the value of δ

$$\delta = H_2(k) \bmod N$$

Chaos with cycles

To fix the issue of cycles pick the table size to be a prime number and also compute the value of δ

$$\delta = H_2(k) \bmod N$$

If $\delta = 0$ then we are guaranteed to be stuck in a cycle, so when this happens set $\delta = 1$

Chaos with cycles

To fix the issue of cycles pick the table size to be a prime number and also compute the value of δ

$$\delta = H_2(k) \bmod N$$

If $\delta = 0$ then we are guaranteed to be stuck in a cycle, so when this happens set $\delta = 1$

Notice that $1 \leq \delta < N$ and **GCD**$(\delta,N) = 1$ since N is prime. Hence, with these conditions we know that modulo N the sequence

$$H_1(k),\ H_1(k)+1\delta,\ H_1(k)+2\delta,\ H_1(k)+3\delta,\ H_1(k)+4\delta,\ \ldots$$

is certain to have order N :)

Constructing H_2(k)

Suppose the key k has type T

Whenever we want to use double hashing as a collision resolution method we need to fabricate a new function H_2(k) that knows how to hash keys of type T.

Constructing H_2(k)

Suppose the key k has type T

Whenever we want to use double hashing as a collision resolution method we need to fabricate a new function H_2(k) that knows how to hash keys of type T.

It would be nice to have a systematic way to be able to effectively produce a new hash function every time we need one, right?

Constructing H_2(k)

Suppose the key k has type T

Whenever we want to use double hashing as a collision resolution method we need to fabricate a new function H_2(k) that knows how to hash keys of type T.

It would be nice to have a systematic way to be able to effectively produce a new hash function every time we need one, right?

Luckily for us the keys we need to hash are always composed of the same fundamental building blocks. In particular: integers, strings, real numbers, fixed length vectors, etc…

Constructing $H_2(k)$

There are many well known high quality hash functions for these fundamental data types. Hence, we can use and combine them to construct our function $H_2(k)$.

Frequently the hash functions selected to compose $H_2(k)$ are picked from a pool of hash functions called **universal hash functions** which generally operate on one fundamental data type.

Inserting with DH

0	1	2	3	4	5	6
∅	∅	∅	∅	∅	∅	∅

Suppose we have an originally empty hash table and we want to insert some (k_i,v_i) pairs with DH and we selected our hash table to have:

Probing function: $P(x) = x*H_2(k)$
Table size: N = 7 (a prime number)
Max load factor: $\alpha = 0.75$
Threshold before resize = $N * \alpha = 5$

0	1	2	3	4	5	6
∅	∅	∅	∅	∅	∅	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6
∅	∅	∅	∅	∅	∅	∅

Suppose $H_1(k_1) = 67$, $H_2(k_1) = 34$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6
∅	∅	∅	∅	∅	∅	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $\mathbf{H_1}(k_1) = 67$, $\mathbf{H_2}(k_1) = 34$

$$\delta = \mathbf{H_2}(k_1) \bmod 7 = 6$$

0	1	2	3	4	5	6
∅	∅	∅	∅	k_1, v_1	∅	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $\mathbf{H_1}(k_1) = 67$, $\mathbf{H_2}(k_1) = 34$

$$\delta = \mathbf{H_2}(k_1) \bmod 7 = 6$$

$$\mathbf{H_1}(k_1) + 0*\delta \bmod 7 = 4$$

0	1	2	3	4	5	6
∅	∅	∅	∅	k_1,v_1	∅	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6
∅	∅	∅	∅	k_1,v_1	∅	∅

Suppose $H_1(k_2) = 2$, $H_2(k_2) = -79$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

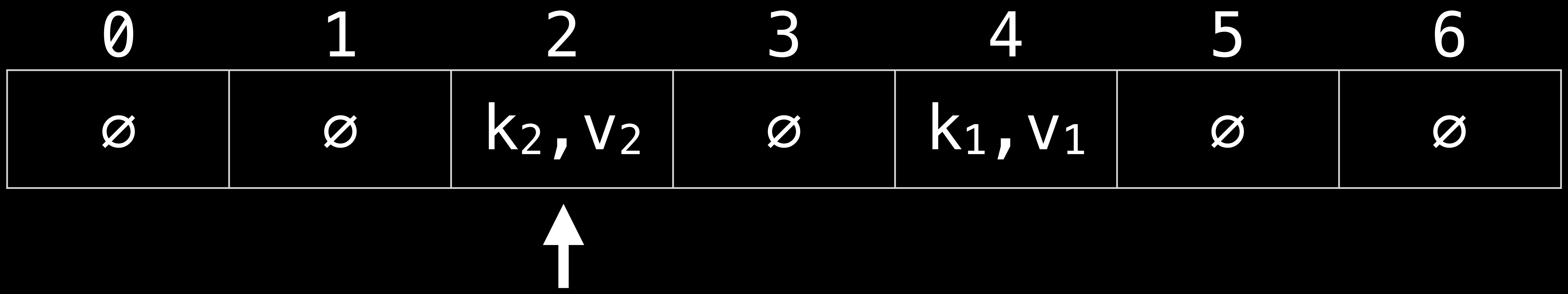
0	1	2	3	4	5	6
∅	∅	∅	∅	k_1,v_1	∅	∅

Suppose $H_1(k_2) = 2$, $H_2(k_2) = -79$

$$\delta = H_2(k_2) \bmod 7 = 5$$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)



0	1	2	3	4	5	6
∅	∅	k_2,v_2	∅	k_1,v_1	∅	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $\mathbf{H_1}(k_2) = 2$, $\mathbf{H_2}(k_2) = -79$

$$\delta = \mathbf{H_2}(k_2) \text{ mod } 7 = 5$$

$$\mathbf{H_1}(k_2) + 0*\delta \text{ mod } 7 = 2$$

0	1	2	3	4	5	6
∅	∅	k_2,v_2	∅	k_1,v_1	∅	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6
∅	∅	k_2, v_2	∅	k_1, v_1	∅	∅

Suppose $H_1(k_3) = 2$, $H_2(k_3) = 10$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

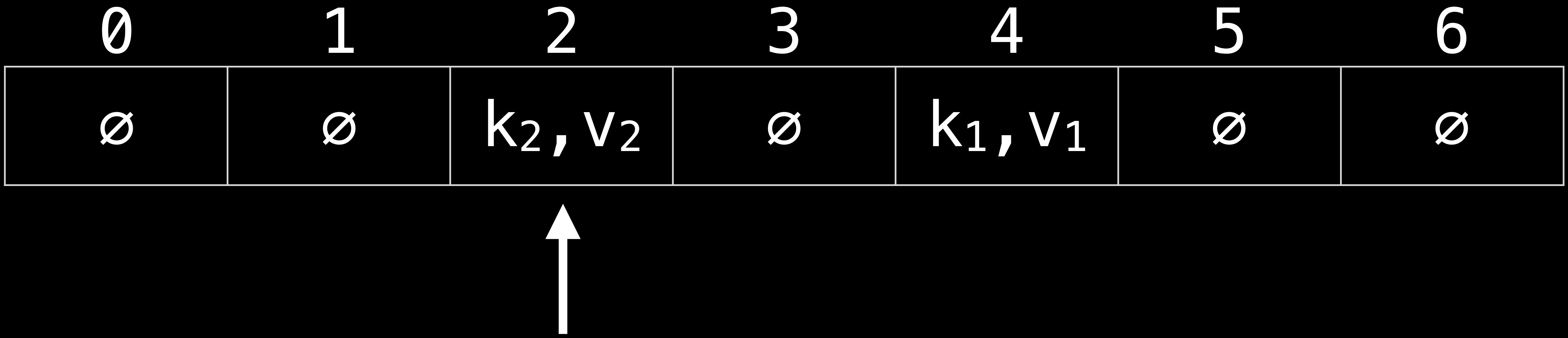
0	1	2	3	4	5	6
∅	∅	k_2,v_2	∅	k_1,v_1	∅	∅

Operations:

$\text{insert}(k_1,v_1)$
$\text{insert}(k_2,v_2)$
$\text{insert}(k_3,v_3)$
$\text{insert}(k_4,v_4)$
$\text{insert}(k_3,v_5)$
$\text{insert}(k_6,v_6)$
$\text{insert}(k_7,v_7)$

Suppose $H_1(k_3) = 2$, $H_2(k_3) = 10$

$$\delta = H_2(k_3) \bmod 7 = 3$$



0	1	2	3	4	5	6
∅	∅	k_2,v_2	∅	k_1,v_1	∅	∅

Suppose $H_1(k_3) = 2$, $H_2(k_3) = 10$

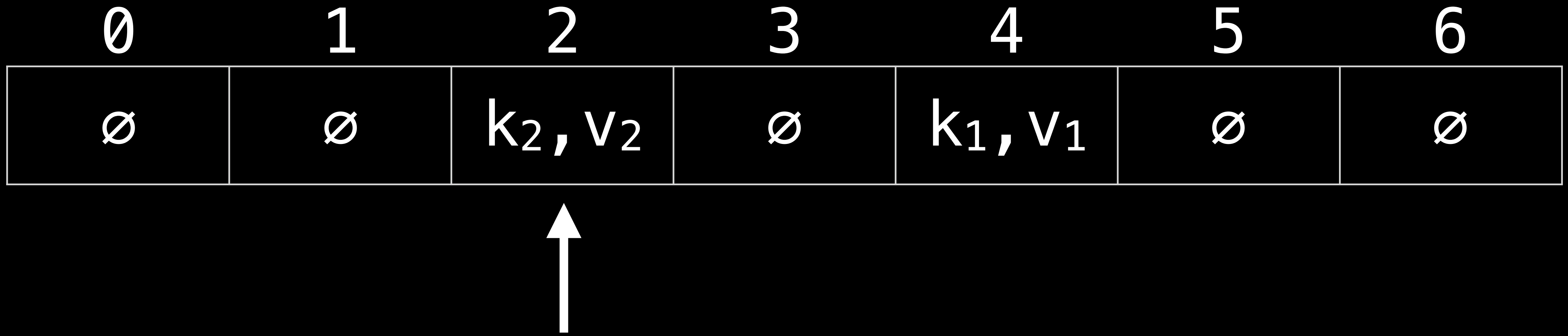
$\delta = H_2(k_3) \bmod 7 = 3$

$H_1(k_3) + 0*\delta \bmod 7 = 2$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Bucket at index 2 is full, so keep probing



0	1	2	3	4	5	6
∅	∅	k_2,v_2	∅	k_1,v_1	∅	∅

Suppose $H_1(k_3) = 2$, $H_2(k_3) = 10$

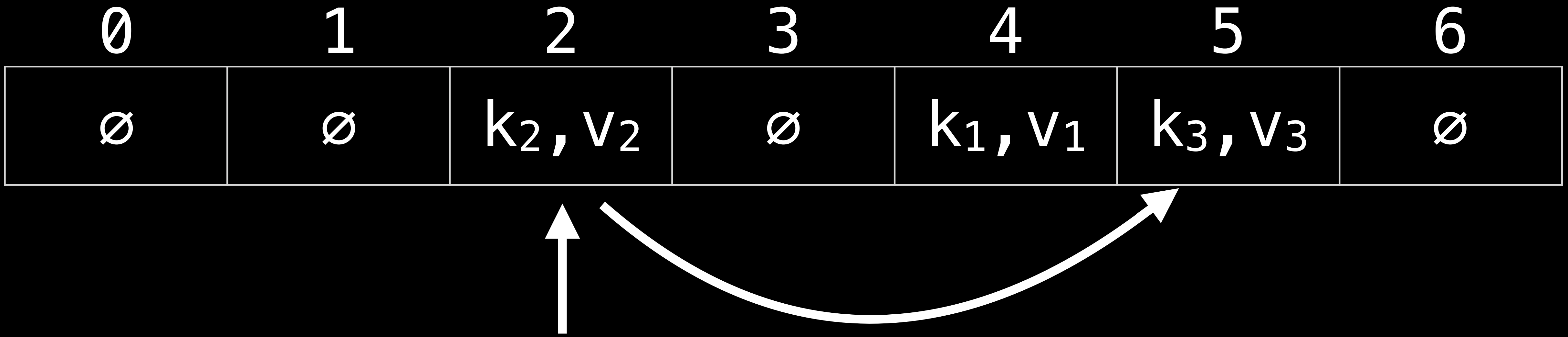
$\delta = H_2(k_3) \bmod 7 = 3$

$H_1(k_3) + 0*\delta \bmod 7 = 2$

$H_1(k_3) + 1*\delta \bmod 7 = 5$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)



0	1	2	3	4	5	6
∅	∅	k_2,v_2	∅	k_1,v_1	k_3,v_3	∅

Operations:

$\text{insert}(k_1,v_1)$
$\text{insert}(k_2,v_2)$
$\text{insert}(k_3,v_3)$
$\text{insert}(k_4,v_4)$
$\text{insert}(k_3,v_5)$
$\text{insert}(k_6,v_6)$
$\text{insert}(k_7,v_7)$

Suppose $H_1(k_3) = 2$, $H_2(k_3) = 10$

$$\delta = H_2(k_3) \bmod 7 = 3$$

$$H_1(k_3) + 0*\delta \bmod 7 = 2$$

$$H_1(k_3) + 1*\delta \bmod 7 = 5$$

0	1	2	3	4	5	6
∅	∅	k_2,v_2	∅	k_1,v_1	k_3,v_3	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6
∅	∅	k_2, v_2	∅	k_1, v_1	k_3, v_3	∅

Suppose $H_1(k_4) = 2$, $H_2(k_4) = 7$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6
∅	∅	k_2,v_2	∅	k_1,v_1	k_3,v_3	∅

Suppose $H_1(k_4) = 2$, $H_2(k_4) = 7$

$\delta = H_2(k_4) \bmod 7 = 0$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6
∅	∅	k_2, v_2	∅	k_1, v_1	k_3, v_3	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H_1(k_4) = 2$, $H_2(k_4) = 7$

$\delta = H_2(k_4) \bmod 7 = 0$

$\delta = 0$, so set $\delta = 1$

0	1	2	3	4	5	6
∅	∅	k_2, v_2	∅	k_1, v_1	k_3, v_3	∅

Suppose $H_1(k_4) = 2$, $H_2(k_4) = 7$

$\delta = 1$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6
∅	∅	k_2,v_2	∅	k_1,v_1	k_3,v_3	∅

Suppose $H_1(k_4) = 2$, $H_2(k_4) = 7$

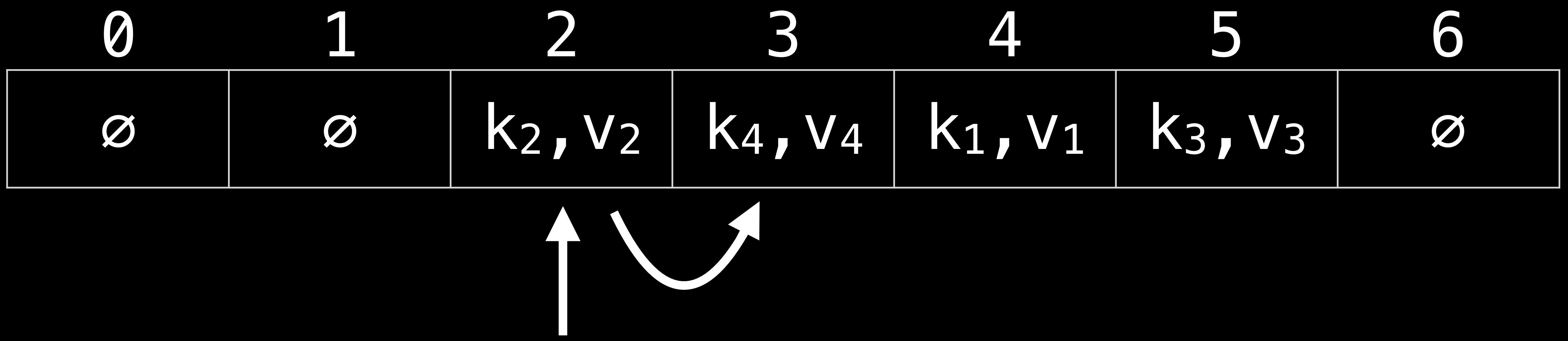
$$\delta = 1$$

$$H_1(k_4) + 0*\delta \bmod 7 = 2$$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Bucket at index 2 is full, so keep probing



0	1	2	3	4	5	6
∅	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_3	∅

Suppose $H_1(k_4) = 2$, $H_2(k_4) = 7$

$$\delta = 1$$

$$H_1(k_4) + 0*\delta \bmod 7 = 2$$
$$H_1(k_4) + 1*\delta \bmod 7 = 3$$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6
∅	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_3	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6
∅	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_3	∅

Suppose $\mathbf{H_1}(k_3) = 2$, $\mathbf{H_2}(k_3) = 10$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6
∅	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_3	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $\mathbf{H_1}(k_3) = 2$, $\mathbf{H_2}(k_3) = 10$

$\delta = \mathbf{H_2}(k_3) \bmod 7 = 3$

0	1	2	3	4	5	6
∅	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_3	∅

↑

Suppose $H_1(k_3) = 2$, $H_2(k_3) = 10$

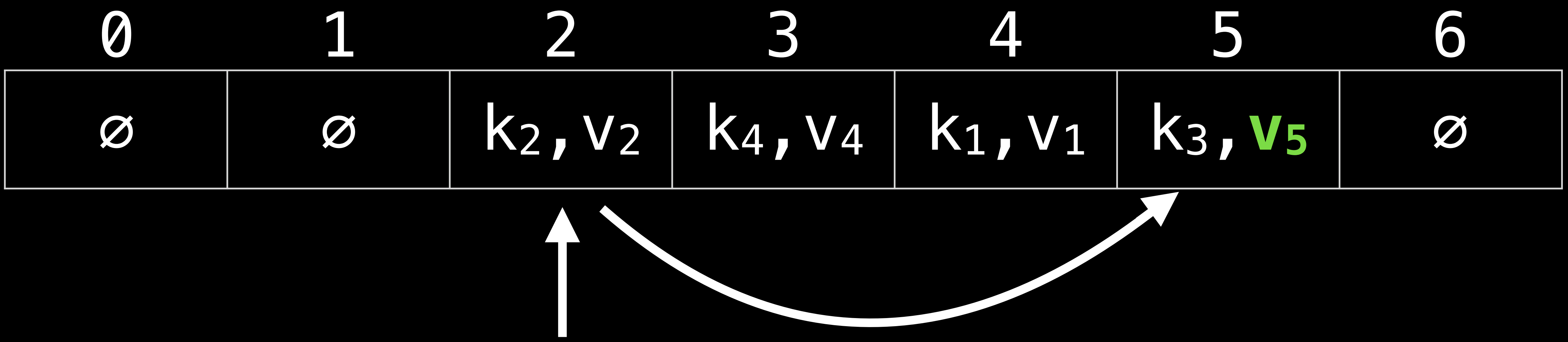
$\delta = H_2(k_3) \bmod 7 = 3$

$H_1(k_3) + 0*\delta \bmod 7 = 2$

Collision at bucket 2

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)



0	1	2	3	4	5	6
∅	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H_1(k_3) = 2$, $H_2(k_3) = 10$

$\delta = H_2(k_3) \bmod 7 = 3$

$H_1(k_3) + 0*\delta \bmod 7 = 2$
$H_1(k_3) + 1*\delta \bmod 7 = 5$

k_3 already existed inside the hash table so update its value

0	1	2	3	4	5	6
∅	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6
∅	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

Suppose $\mathbf{H_1}(k_6) = 3$, $\mathbf{H_2}(k_6) = 23$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

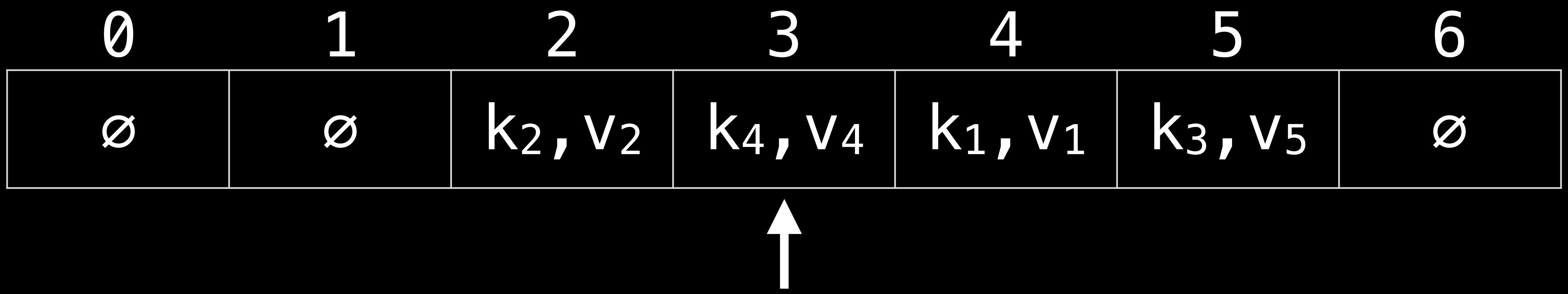
0	1	2	3	4	5	6
∅	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H_1(k_6) = 3$, $H_2(k_6) = 23$

$\delta = H_2(k_6) \bmod 7 = 2$



0	1	2	3	4	5	6
∅	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

Operations:

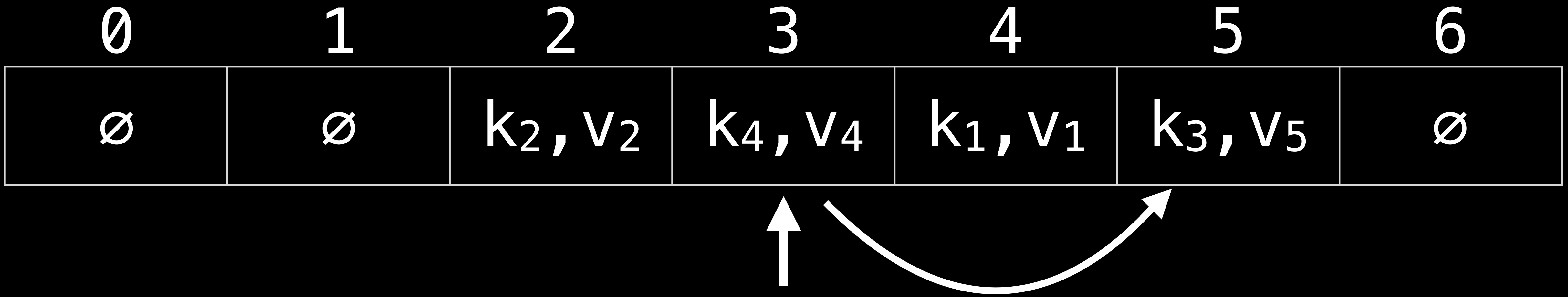
insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H_1(k_6) = 3$, $H_2(k_6) = 23$

$\delta = H_2(k_6) \bmod 7 = 2$

$H_1(k_6) + 0*\delta \bmod 7 = 3$

Bucket at index 3 is full
so keep probing!



0	1	2	3	4	5	6
∅	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

Operations:

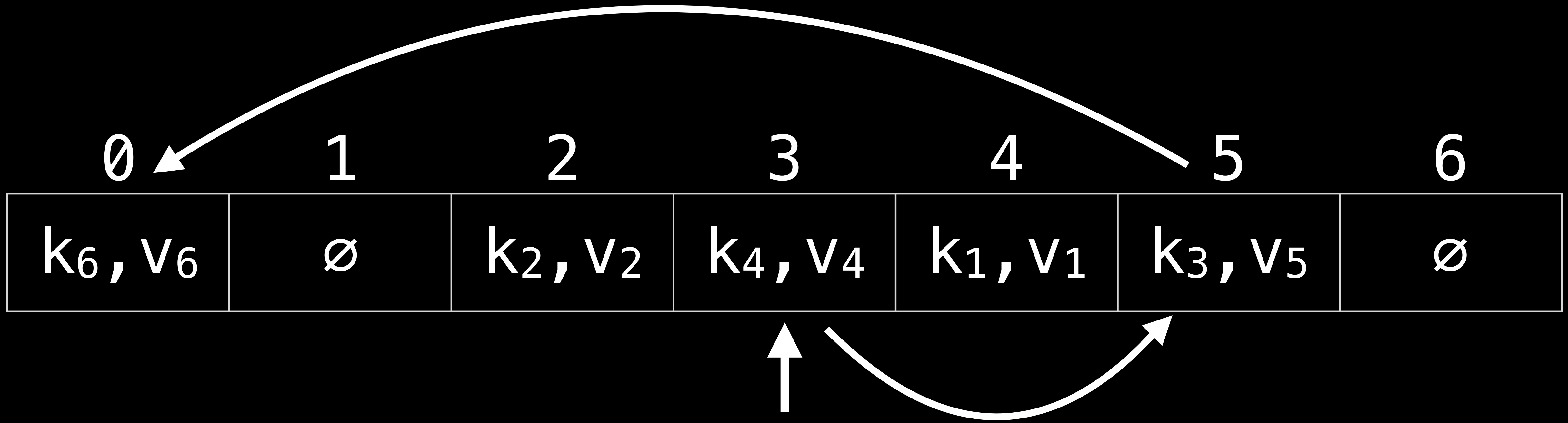
insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $\mathbf{H_1}(k_6) = 3$, $\mathbf{H_2}(k_6) = 23$

$$\delta = \mathbf{H_2}(k_6) \bmod 7 = 2$$

$$\mathbf{H_1}(k_6) + 0*\delta \bmod 7 = 3$$
$$\mathbf{H_1}(k_6) + 1*\delta \bmod 7 = 5$$

Bucket at index 5 is full
so keep probing!



0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H_1(k_6) = 3$, $H_2(k_6) = 23$

$\delta = H_2(k_6) \bmod 7 = 2$

$H_1(k_6) + 0*\delta \bmod 7 = 3$

$H_1(k_6) + 1*\delta \bmod 7 = 5$

$H_1(k_6) + 2*\delta \bmod 7 = 0$

0	1	2	3	4	5	6
k_6,v_6	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

The maximum threshold on this table was five key–value pairs, so it’s time to resize.

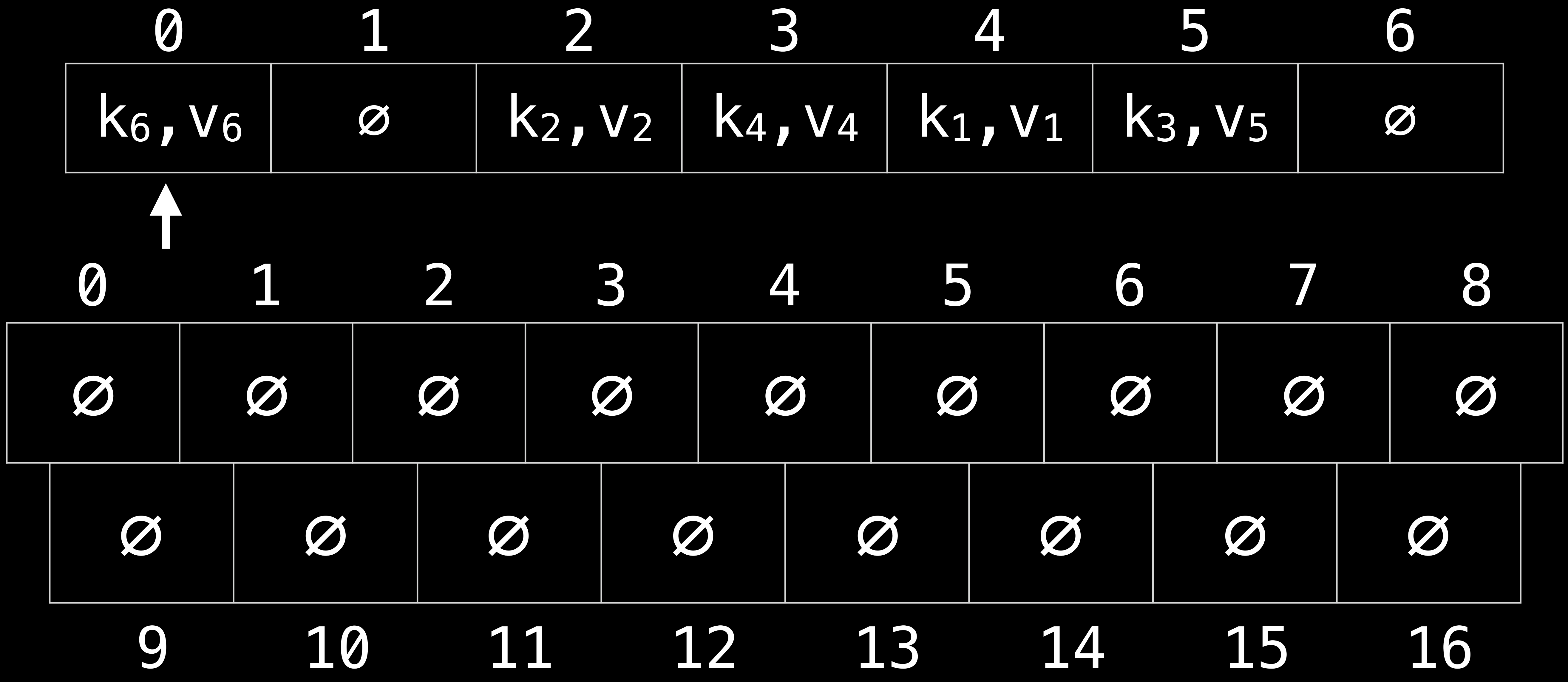
To resize one strategy is compute 2N and find the next prime above this value.

In this case 2N = 14 and the next prime above 14 is 17, so 17 is the new table size.

0	1	2	3	4	5	6
k_6,v_6	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	∅	∅	∅	∅	∅	∅	∅

9	10	11	12	13	14	15	16
∅	∅	∅	∅	∅	∅	∅	∅

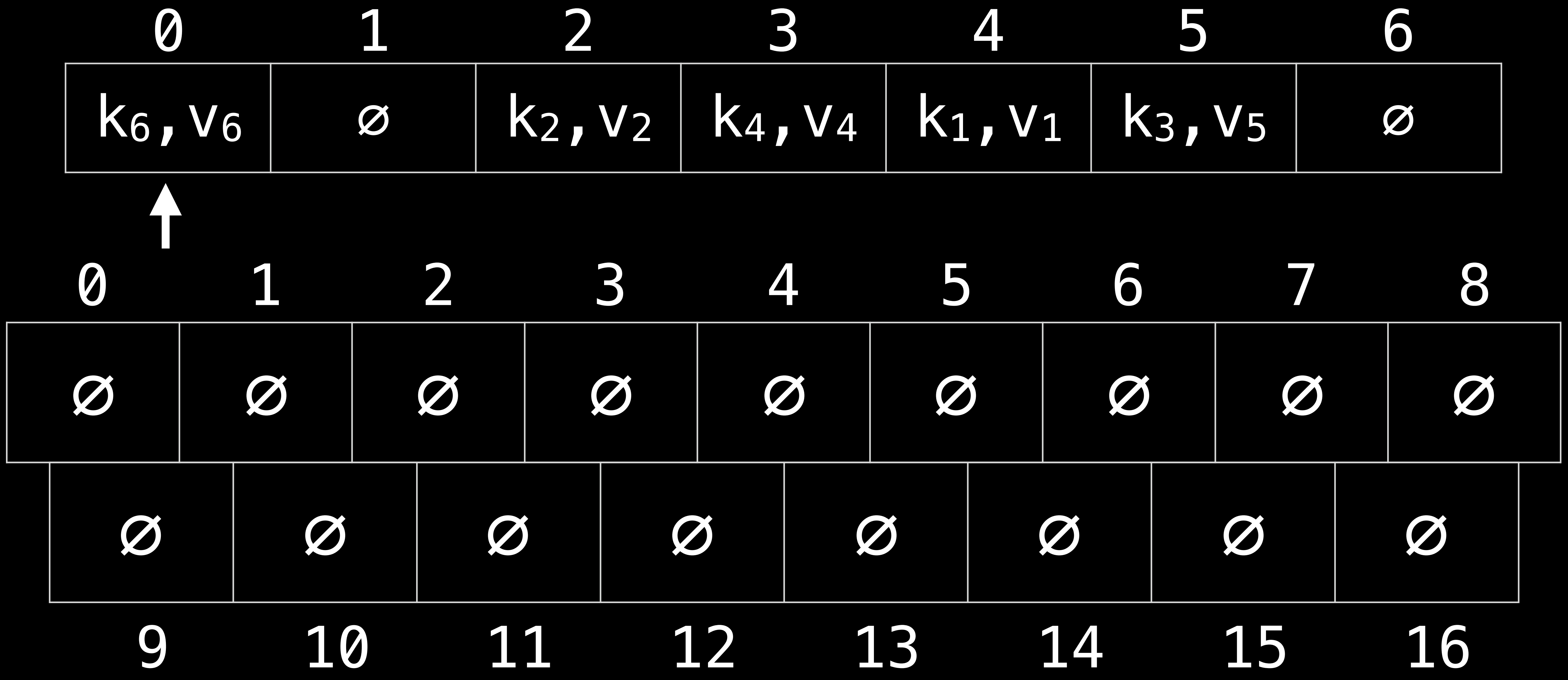


0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	∅	∅	∅	∅	∅	∅	∅

9	10	11	12	13	14	15	16
∅	∅	∅	∅	∅	∅	∅	∅

From before, **H_1**$(k_6) = 3$, **H_2**$(k_6) = 23$

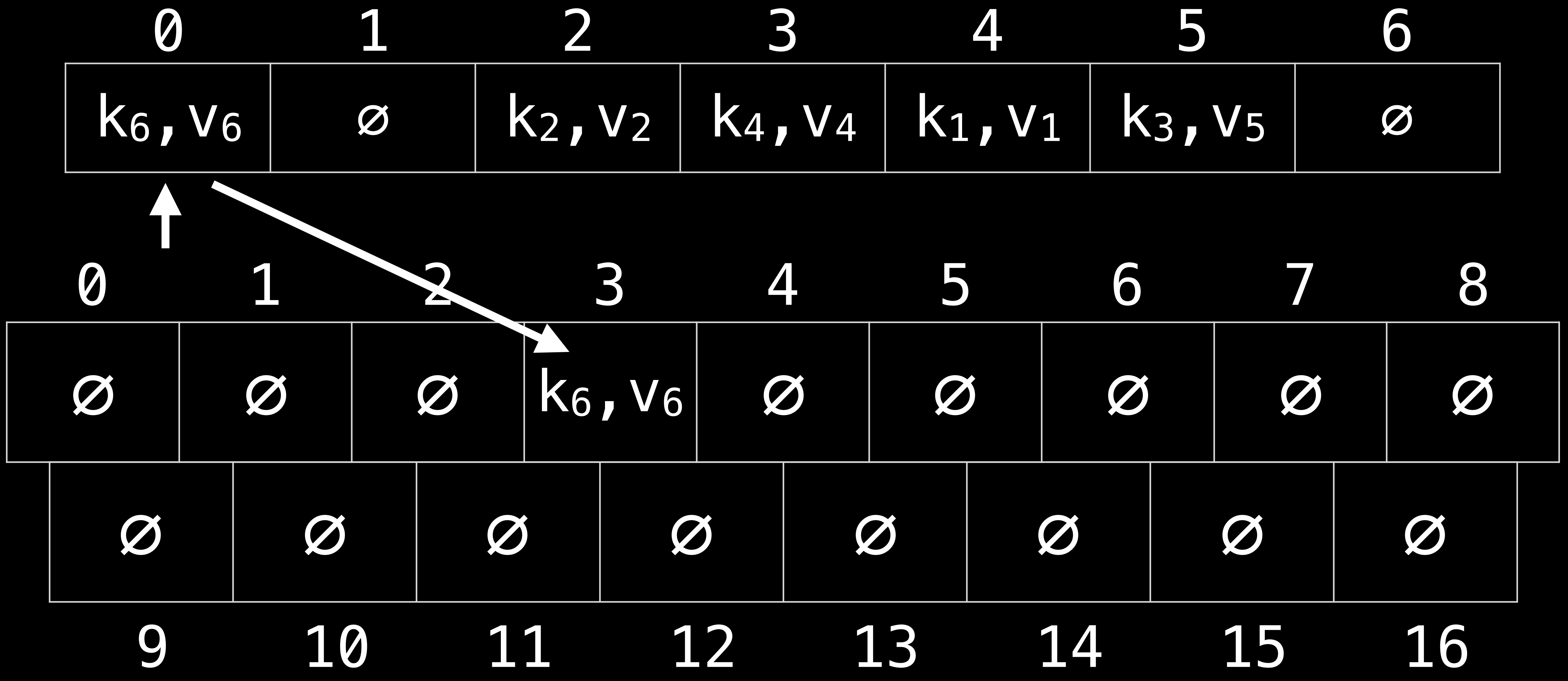


0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅

From before, $\mathbf{H_1}(k_6) = 3$, $\mathbf{H_2}(k_6) = 23$

$\delta = \mathbf{H_2}(k_6) \bmod 17 = 6$



0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

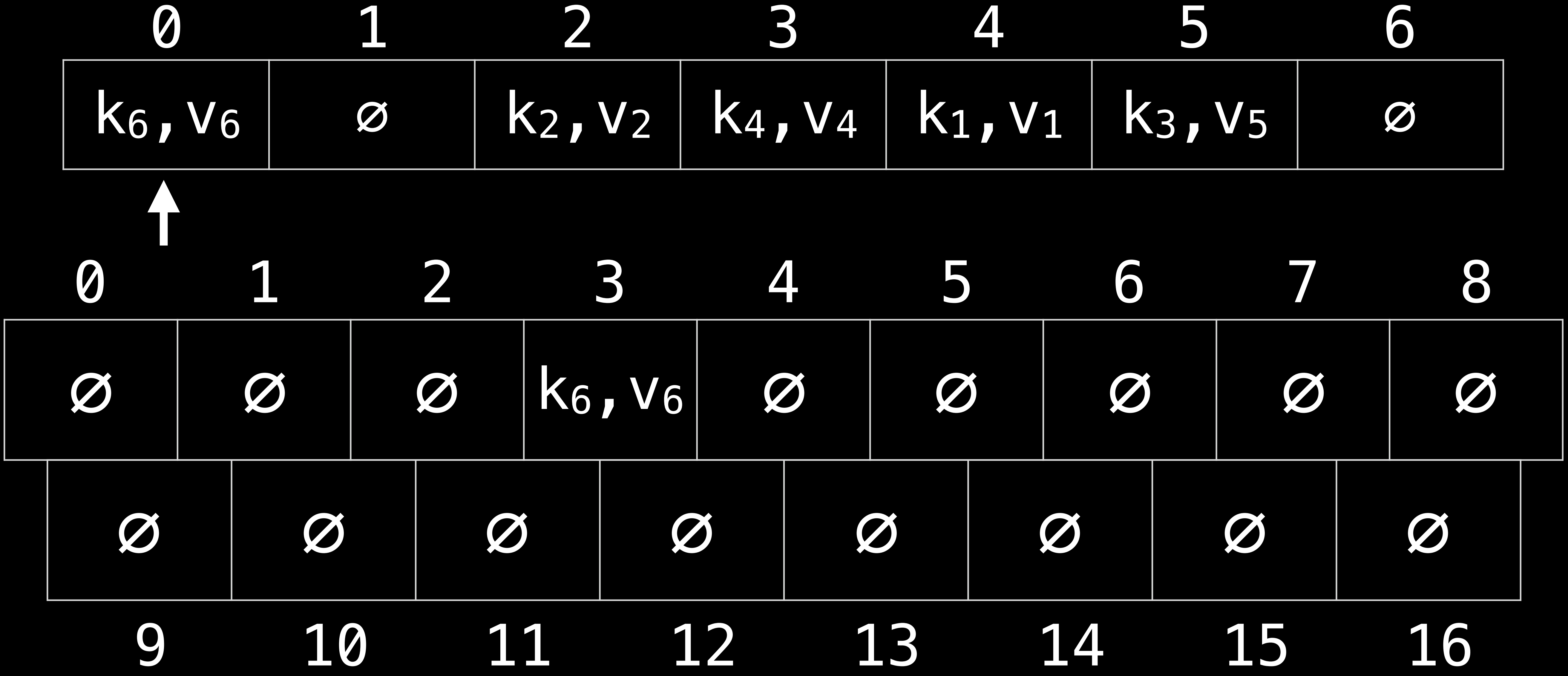
0	1	2	3	4	5	6	7	8
∅	∅	∅	k_6, v_6	∅	∅	∅	∅	∅

9	10	11	12	13	14	15	16
∅	∅	∅	∅	∅	∅	∅	∅

From before, $\mathbf{H_1}(k_6) = 3$, $\mathbf{H_2}(k_6) = 23$

$\delta = \mathbf{H_2}(k_6) \bmod 17 = 6$

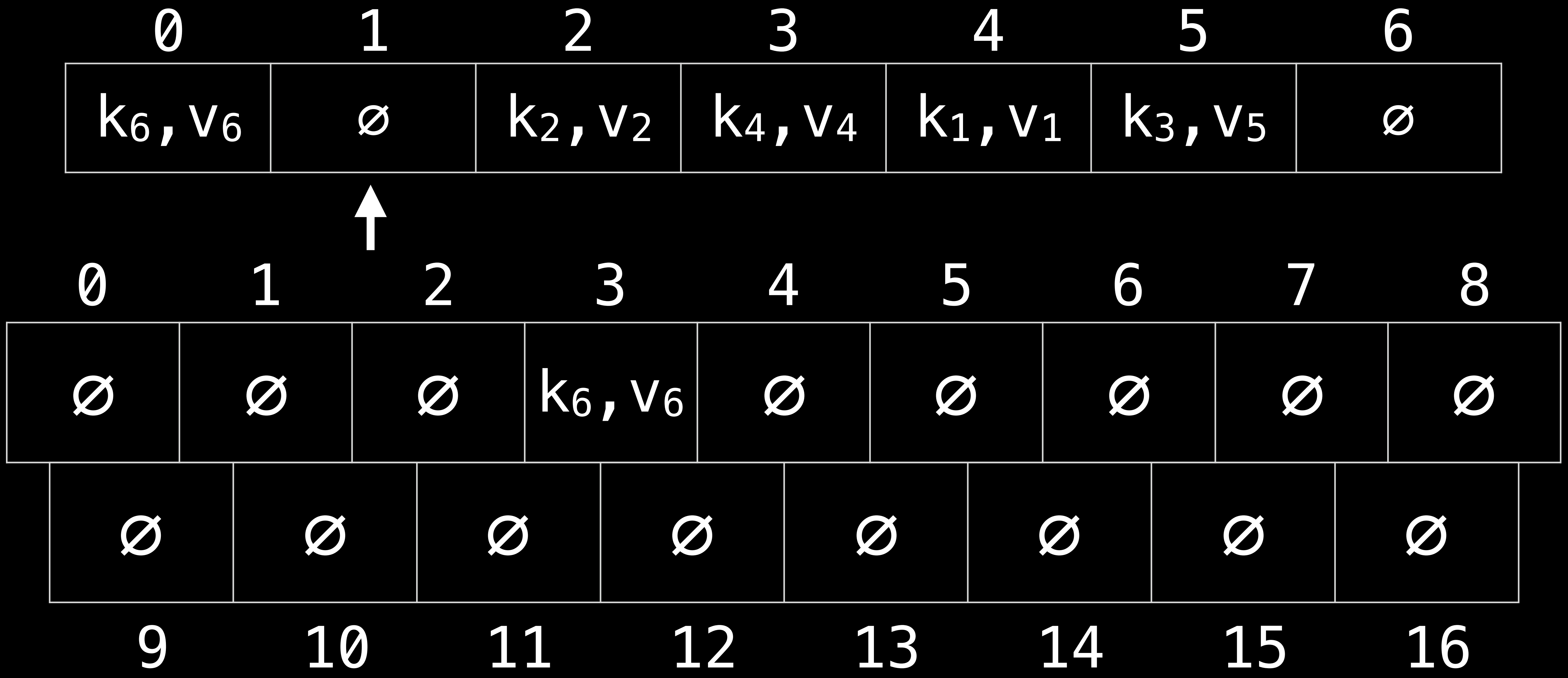
$\mathbf{H_1}(k_6) + 0*\delta \bmod 17 = 3$



0	1	2	3	4	5	6
k_6,v_6	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	∅	k_6,v_6	∅	∅	∅	∅	∅

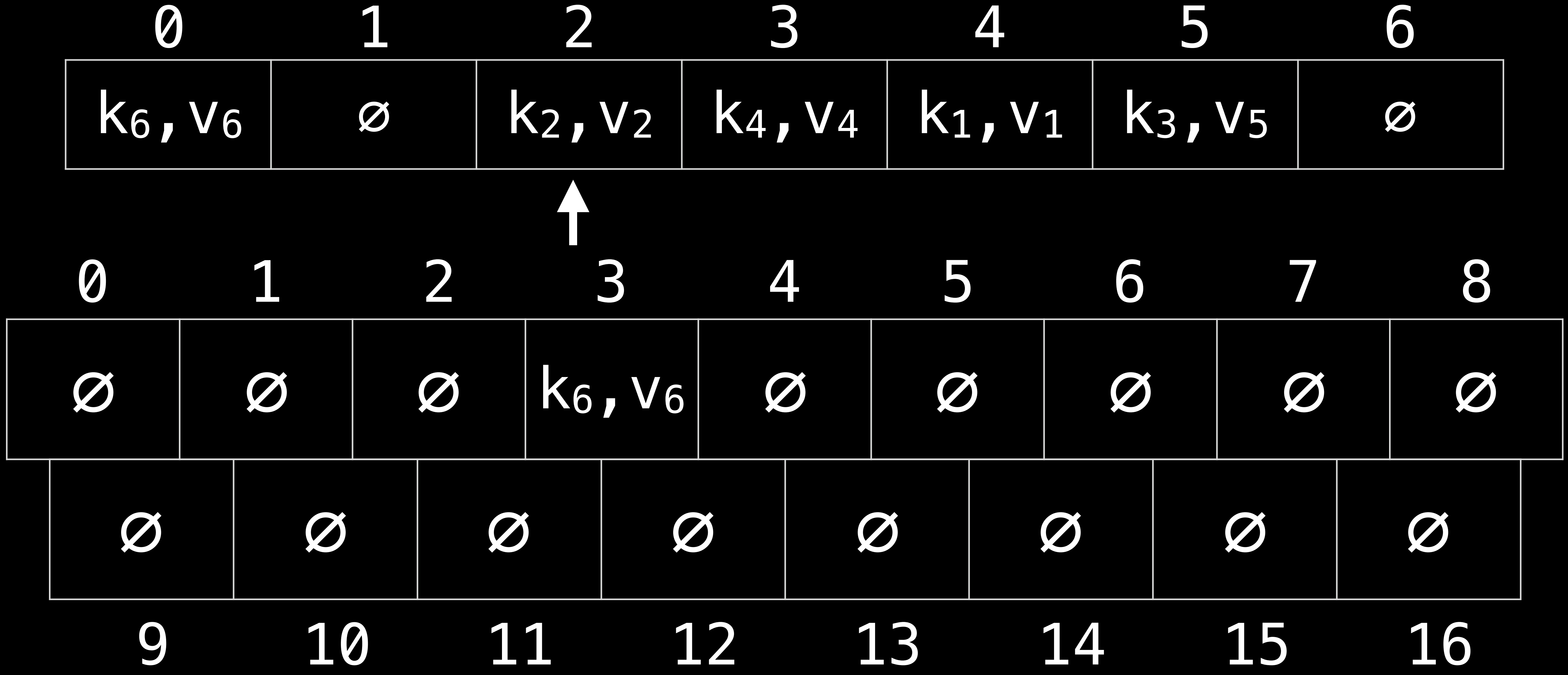
9	10	11	12	13	14	15	16
∅	∅	∅	∅	∅	∅	∅	∅



0	1	2	3	4	5	6
k_6,v_6	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	∅	k_6,v_6	∅	∅	∅	∅	∅

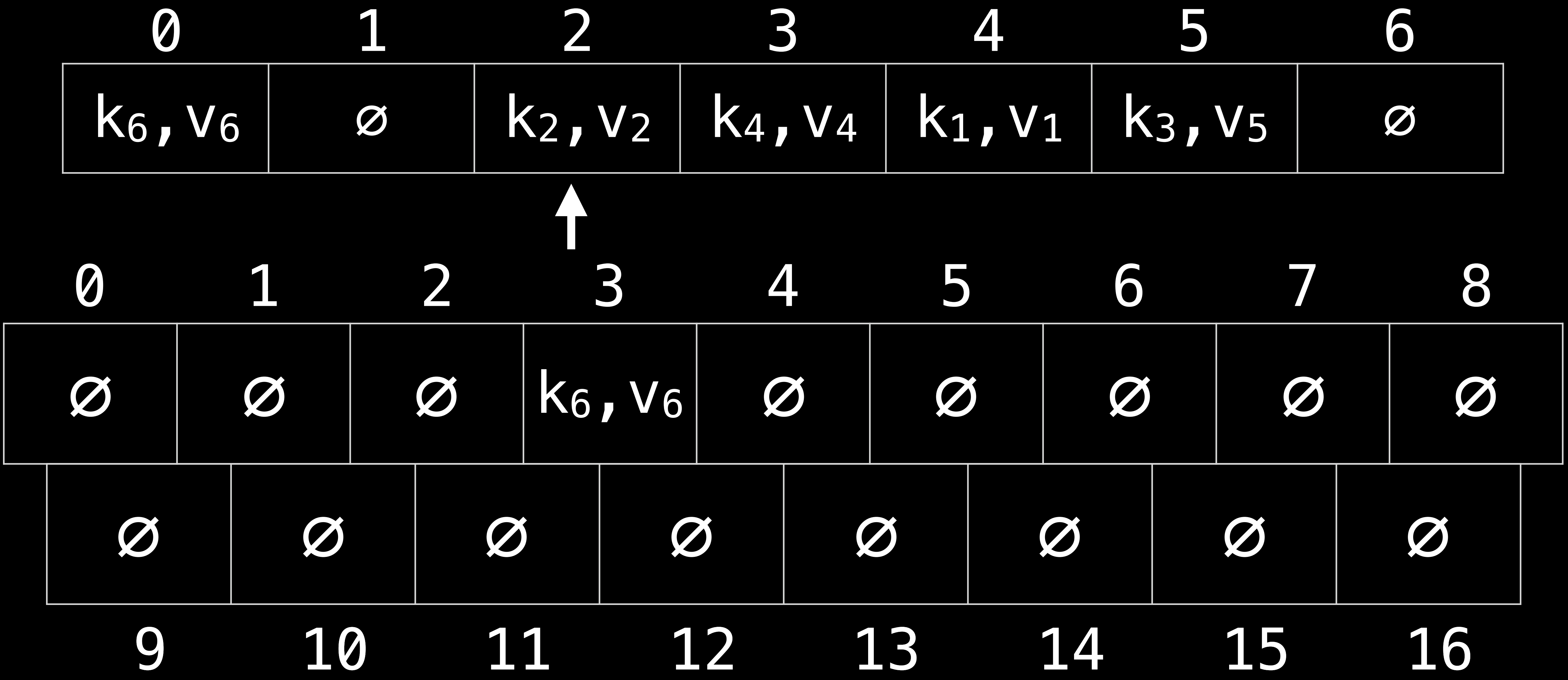
9	10	11	12	13	14	15	16
∅	∅	∅	∅	∅	∅	∅	∅



0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	∅	k_6, v_6	∅	∅	∅	∅	∅

9	10	11	12	13	14	15	16
∅	∅	∅	∅	∅	∅	∅	∅

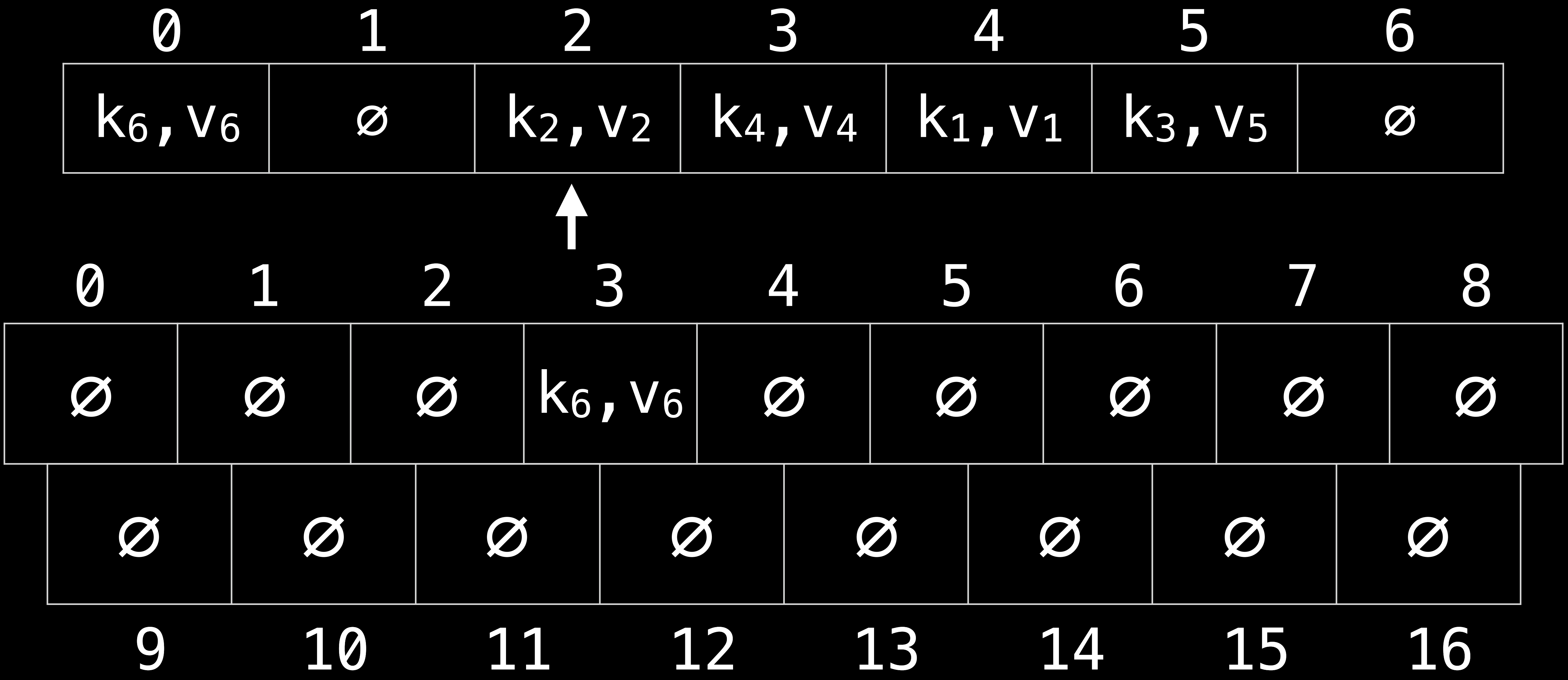


0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	∅	k_6, v_6	∅	∅	∅	∅	∅

9	10	11	12	13	14	15	16
∅	∅	∅	∅	∅	∅	∅	∅

From before, $H_1(k_2) = 2$, $H_2(k_2) = -79$



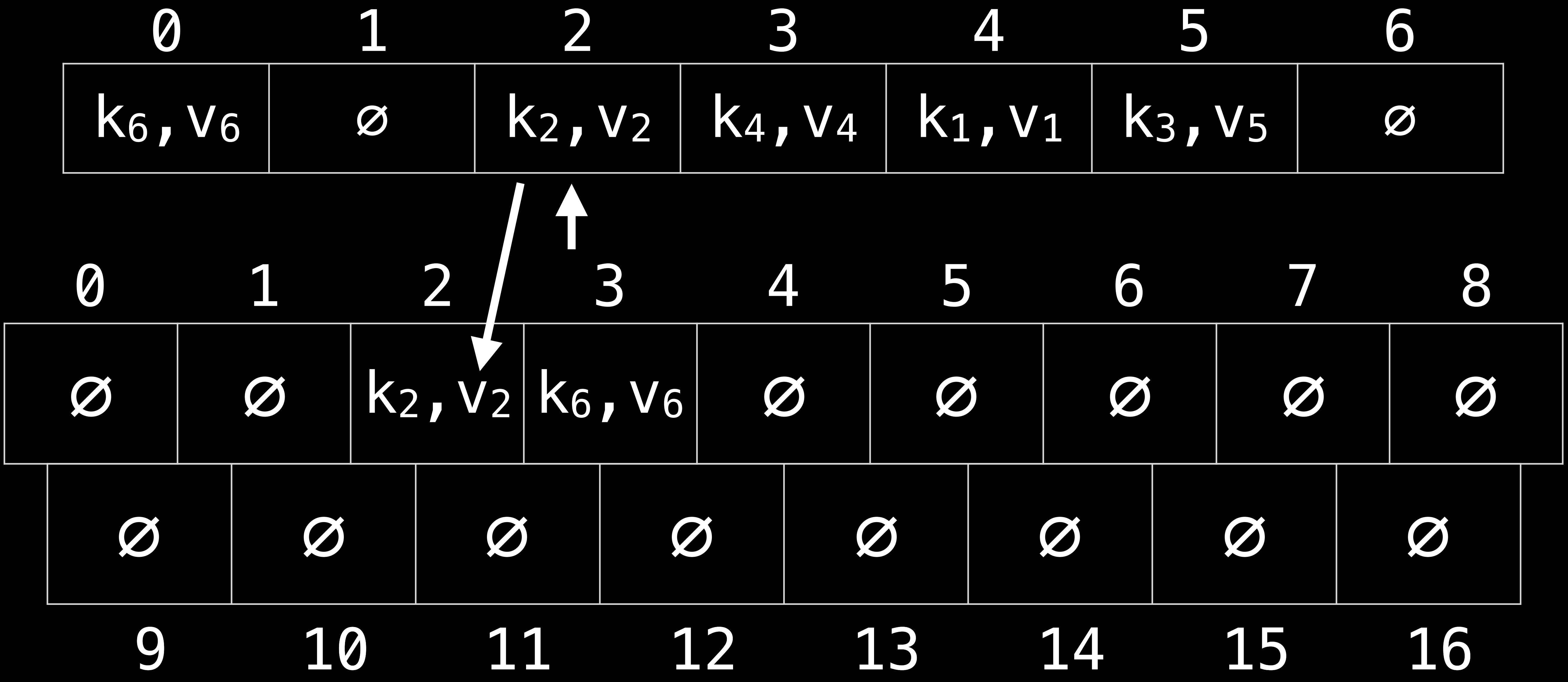
0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	∅	k_6, v_6	∅	∅	∅	∅	∅

9	10	11	12	13	14	15	16
∅	∅	∅	∅	∅	∅	∅	∅

From before, $\mathbf{H_1}(k_2) = 2$, $\mathbf{H_2}(k_2) = -79$

$$\delta = \mathbf{H_2}(k_2) \bmod 17 = 6$$



0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

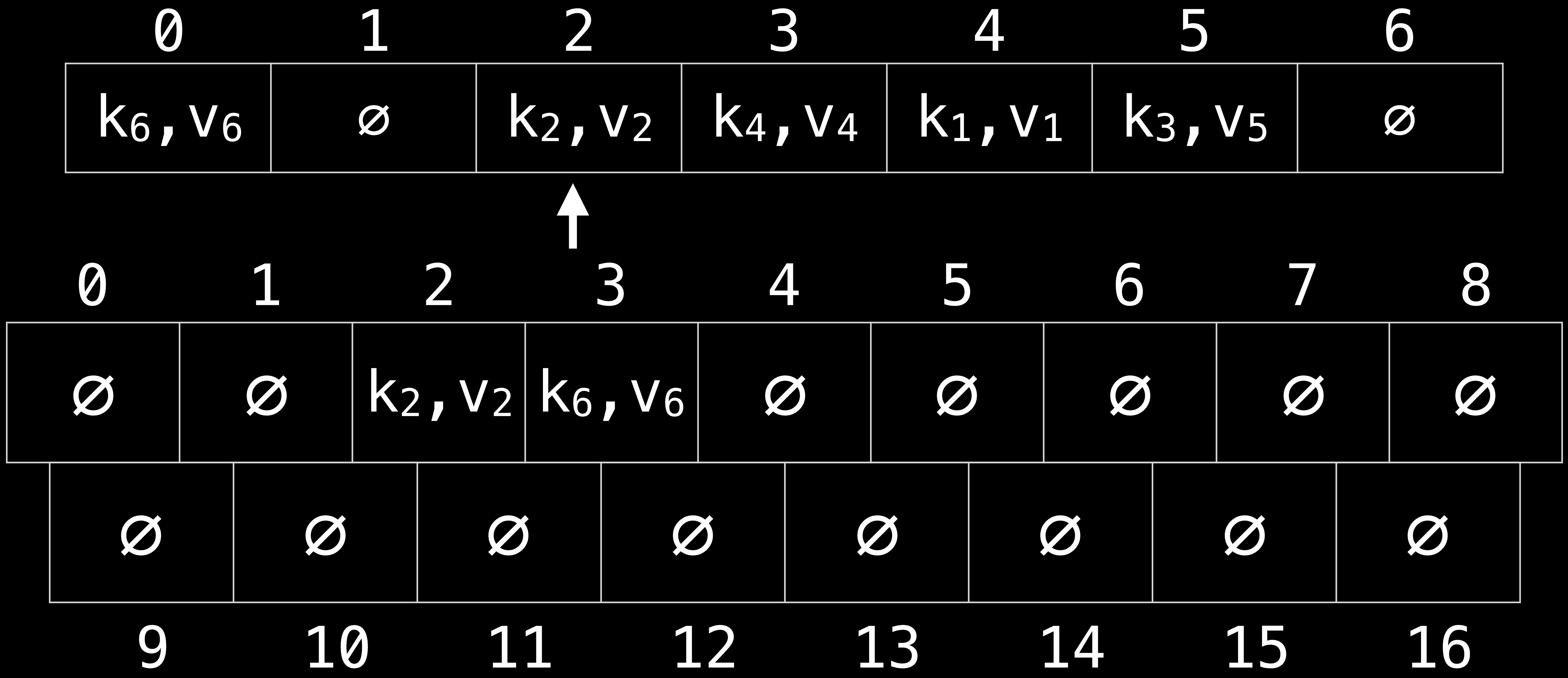
0	1	2	3	4	5	6	7	8
∅	∅	k_2, v_2	k_6, v_6	∅	∅	∅	∅	∅

9	10	11	12	13	14	15	16
∅	∅	∅	∅	∅	∅	∅	∅

From before, $\mathbf{H_1}(k_2) = 2$, $\mathbf{H_2}(k_2) = -79$

$$\delta = \mathbf{H_2}(k_2) \bmod 17 = 6$$

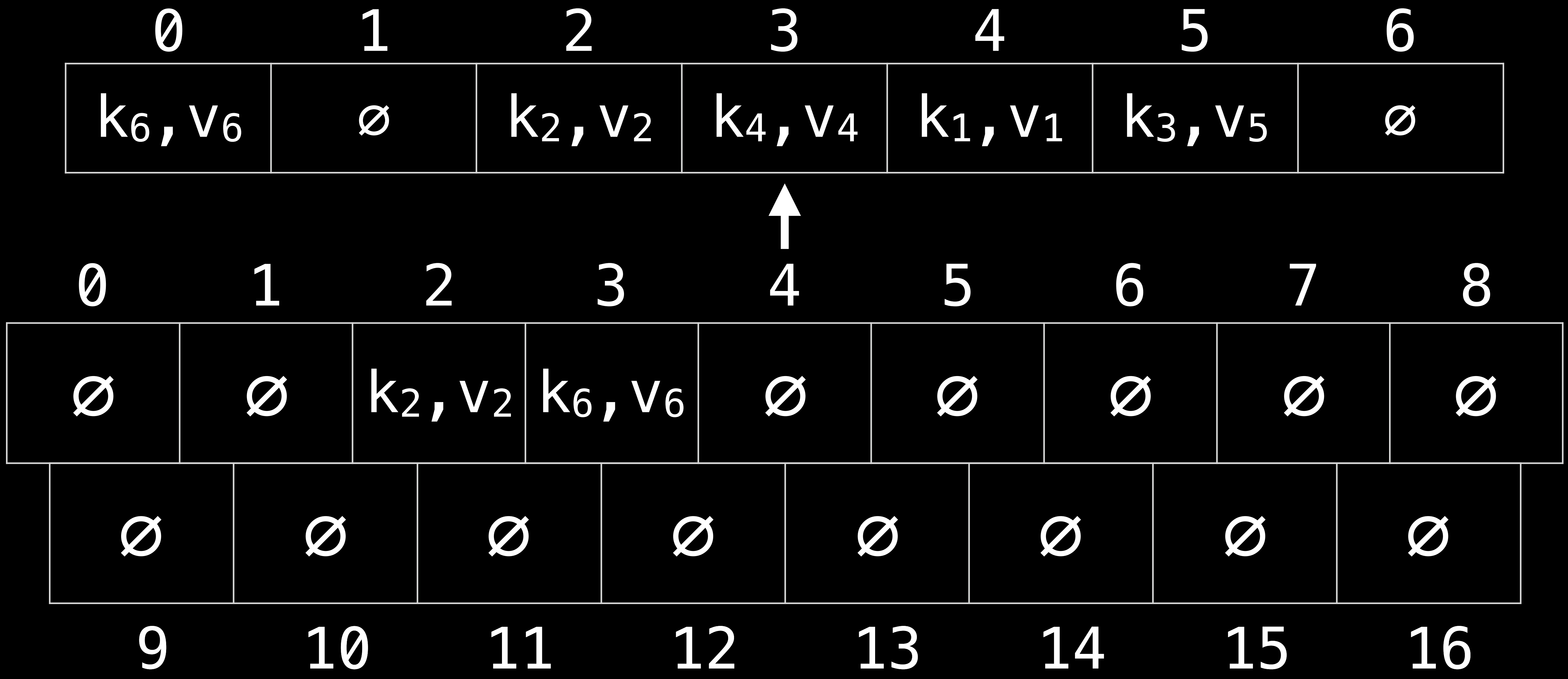
$$\mathbf{H_1}(k_2) + 0 * \delta \bmod 17 = 2$$



0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2, v_2	k_6, v_6	∅	∅	∅	∅	∅

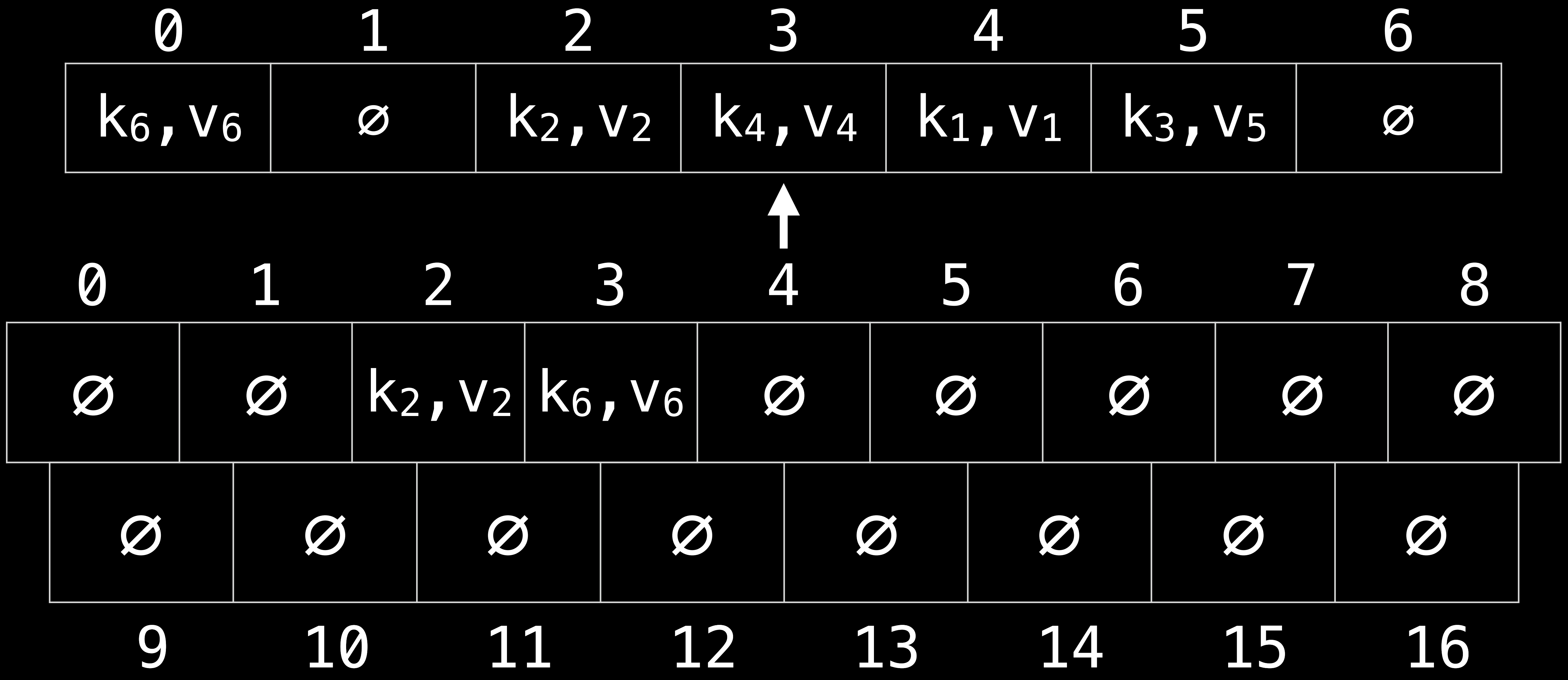
9	10	11	12	13	14	15	16
∅	∅	∅	∅	∅	∅	∅	∅



0	1	2	3	4	5	6
k_6,v_6	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2,v_2	k_6,v_6	∅	∅	∅	∅	∅

9	10	11	12	13	14	15	16
∅	∅	∅	∅	∅	∅	∅	∅

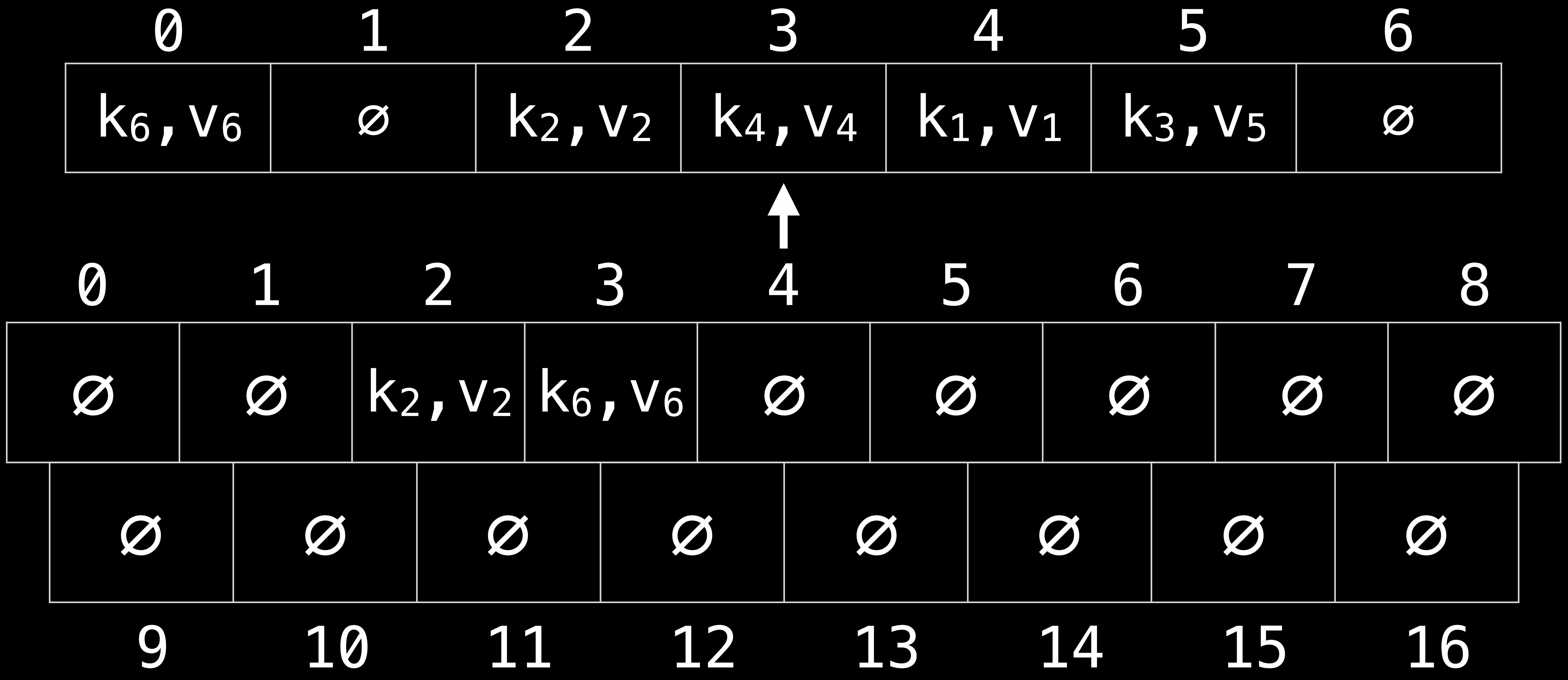


0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2, v_2	k_6, v_6	∅	∅	∅	∅	∅

∅	∅	∅	∅	∅	∅	∅	∅
9	10	11	12	13	14	15	16

From before, $\mathbf{H_1}(k_4) = 2$, $\mathbf{H_2}(k_4) = 7$



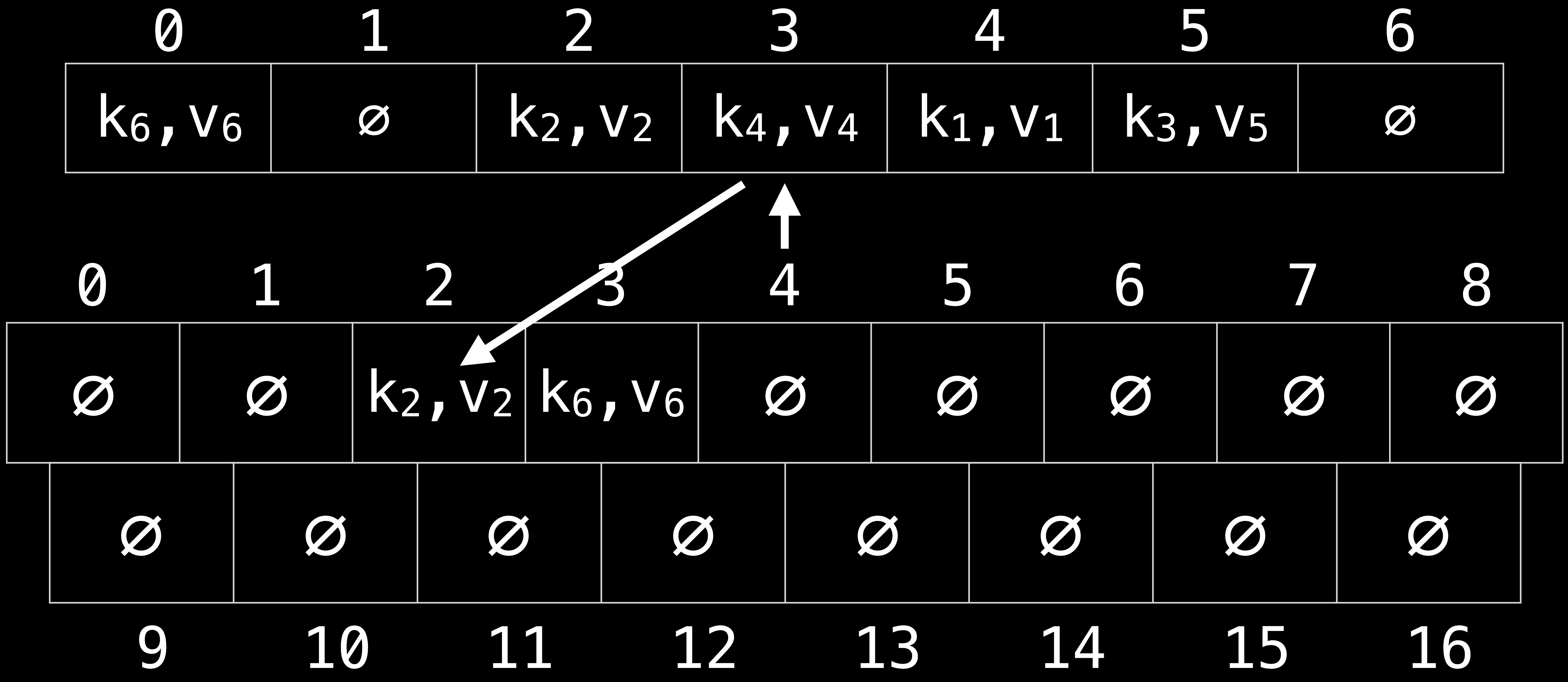
0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2, v_2	k_6, v_6	∅	∅	∅	∅	∅

9	10	11	12	13	14	15	16
∅	∅	∅	∅	∅	∅	∅	∅

From before, $H_1(k_4) = 2$, $H_2(k_4) = 7$

$$\delta = H_2(k_4) \bmod 17 = 7$$



0	1	2	3	4	5	6
k_6,v_6	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2,v_2	k_6,v_6	∅	∅	∅	∅	∅

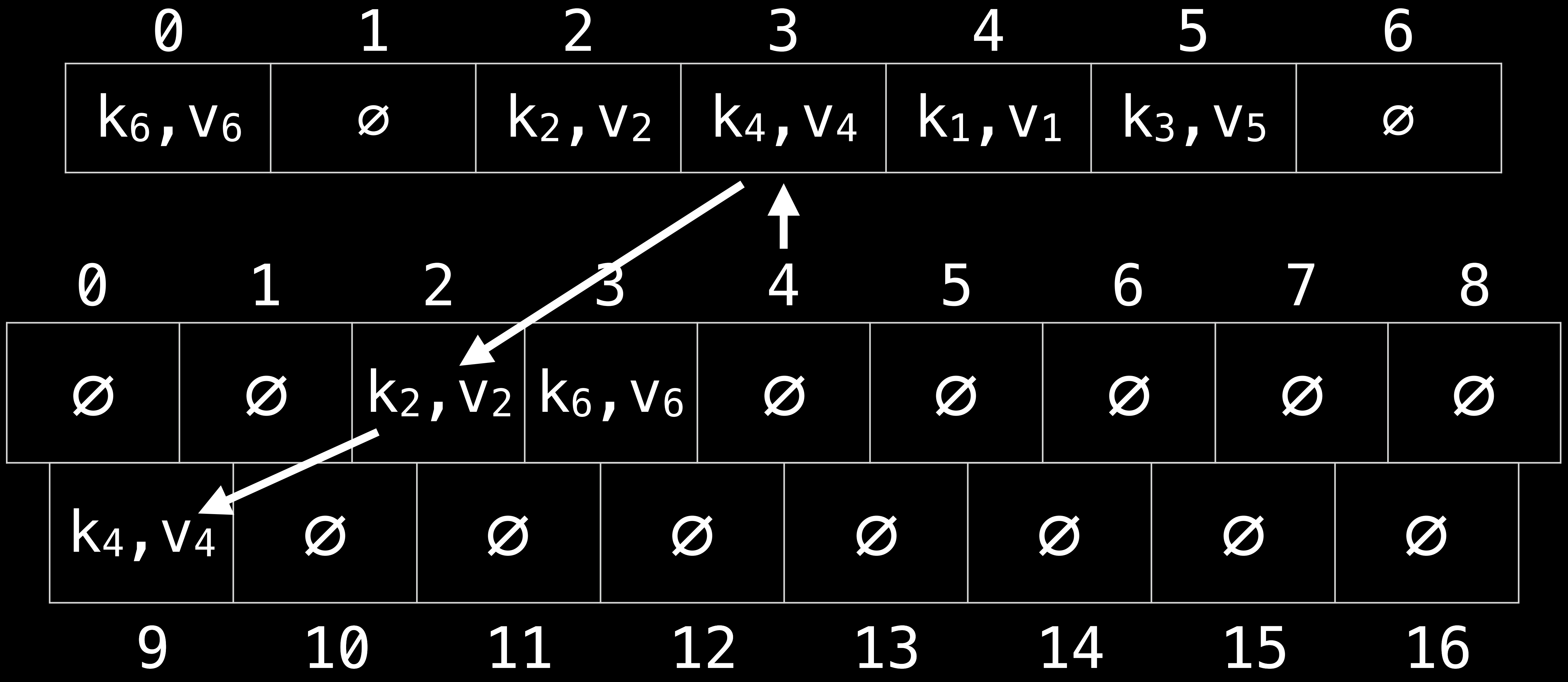
9	10	11	12	13	14	15	16
∅	∅	∅	∅	∅	∅	∅	∅

From before, $\mathbf{H_1}(k_4) = 2$, $\mathbf{H_2}(k_4) = 7$

$$\delta = \mathbf{H_2}(k_4) \bmod 17 = 7$$

$$\mathbf{H_1}(k_4) + 0*\delta \bmod 17 = 2$$

Collision at bucket 2 so keep probing



0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2, v_2	k_6, v_6	∅	∅	∅	∅	∅

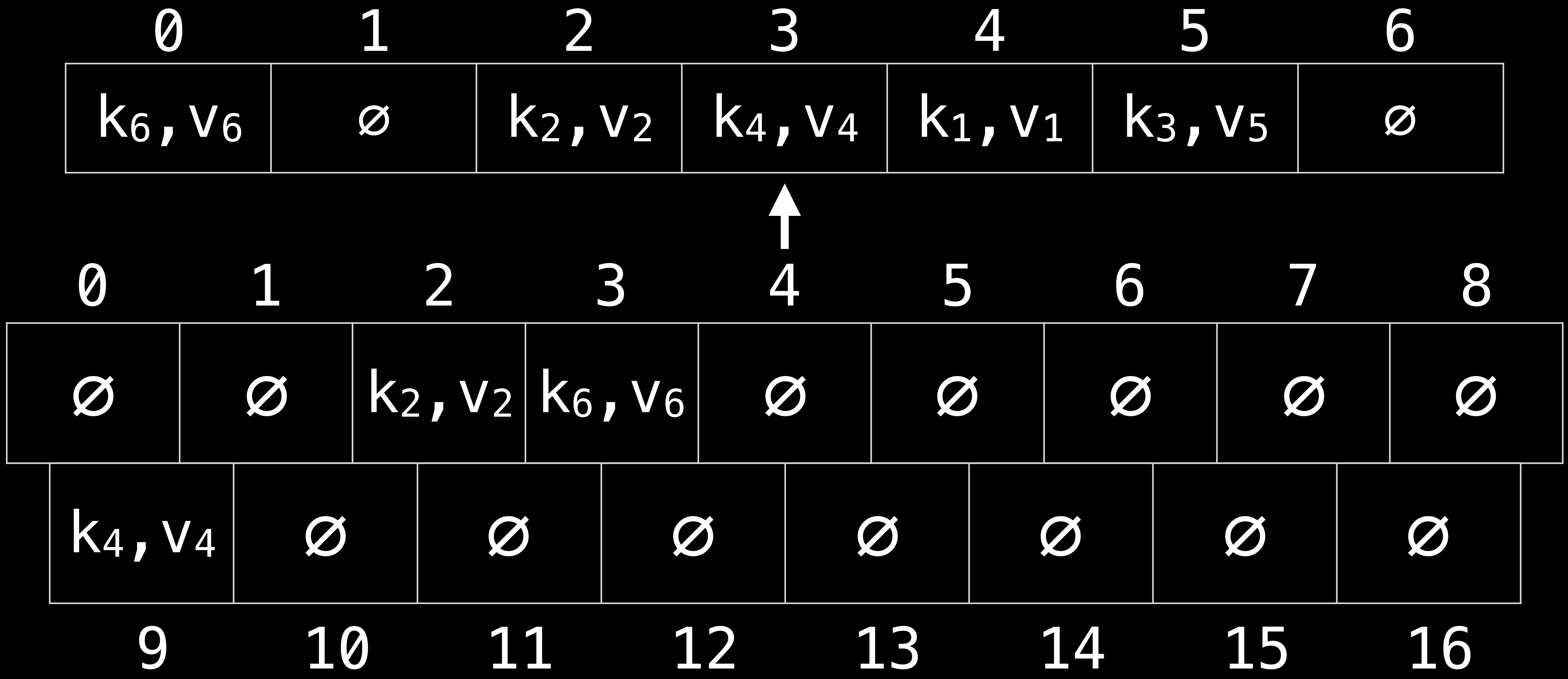
9	10	11	12	13	14	15	16
k_4, v_4	∅	∅	∅	∅	∅	∅	∅

From before, $\mathbf{H_1}(k_4) = 2$, $\mathbf{H_2}(k_4) = 7$

$$\delta = \mathbf{H_2}(k_4) \bmod 17 = 7$$

$$\mathbf{H_1}(k_4) + 0*\delta \bmod 17 = 2$$

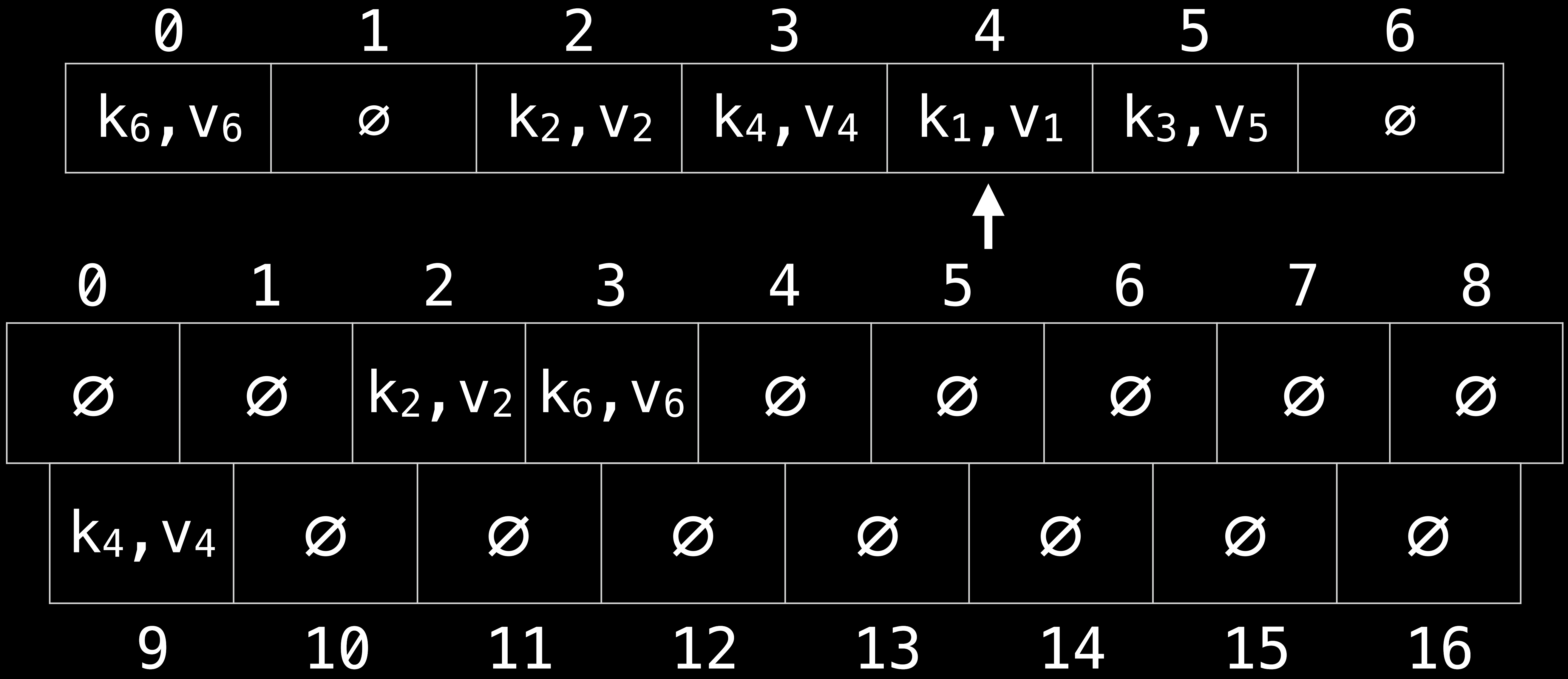
$$\mathbf{H_1}(k_4) + 1*\delta \bmod 17 = 9$$



0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2, v_2	k_6, v_6	∅	∅	∅	∅	∅

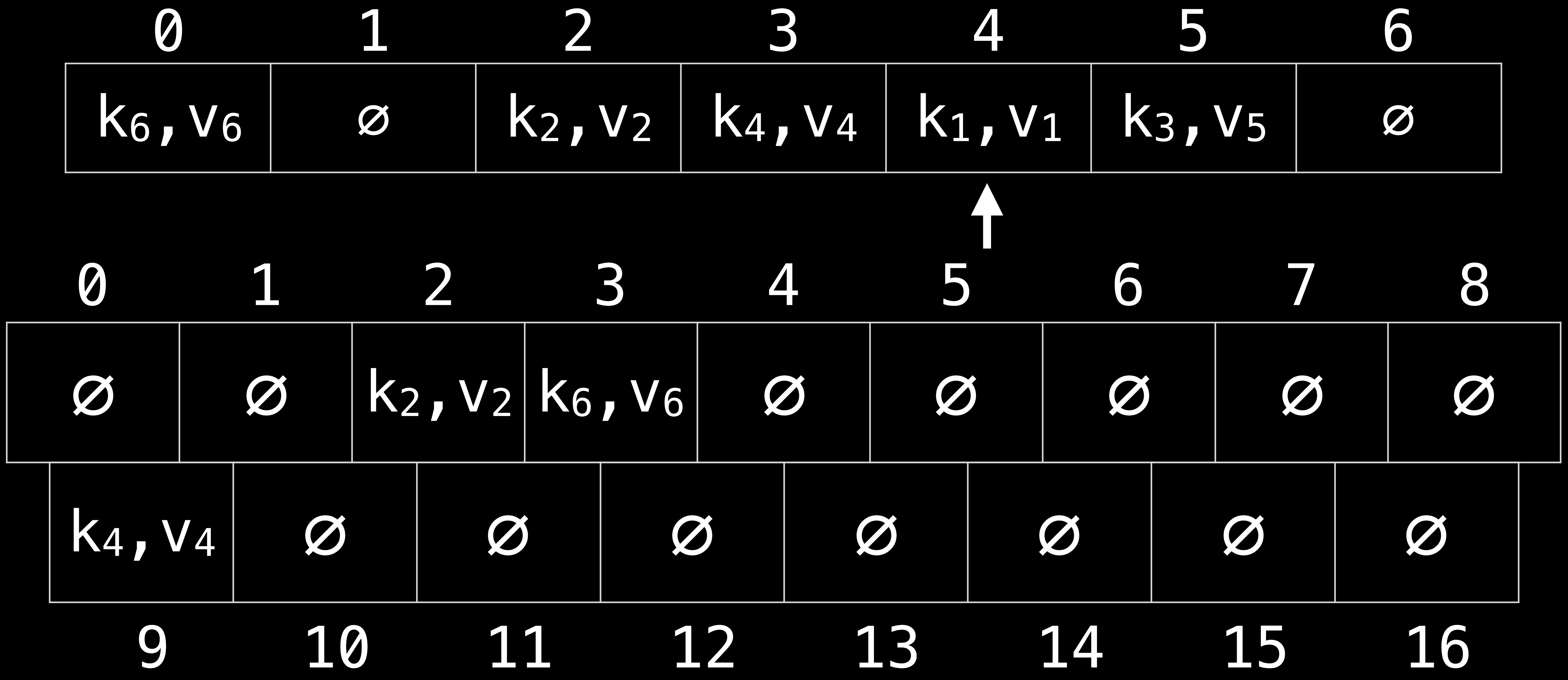
9	10	11	12	13	14	15	16
k_4, v_4	∅	∅	∅	∅	∅	∅	∅

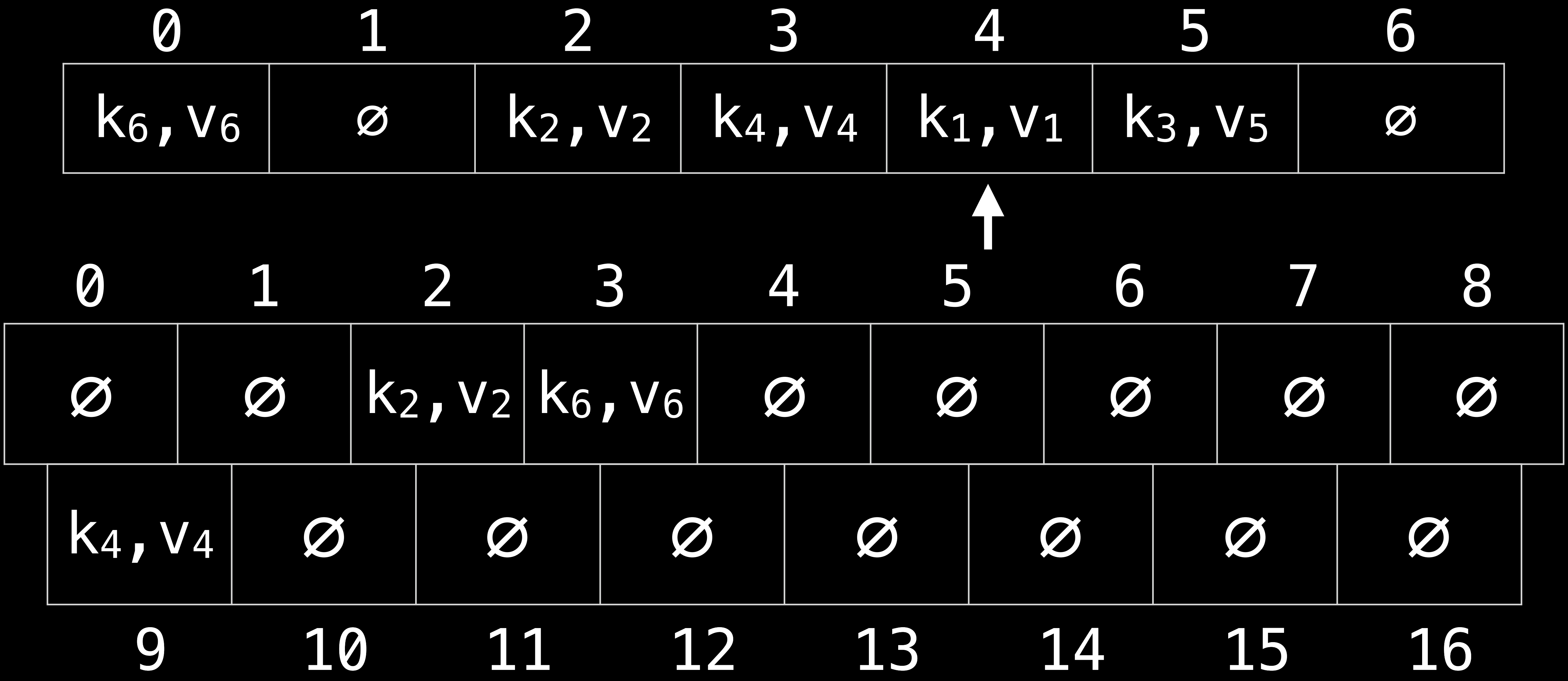


0	1	2	3	4	5	6
k_6,v_6	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2,v_2	k_6,v_6	∅	∅	∅	∅	∅

k_4,v_4	∅	∅	∅	∅	∅	∅	∅
9	10	11	12	13	14	15	16





0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

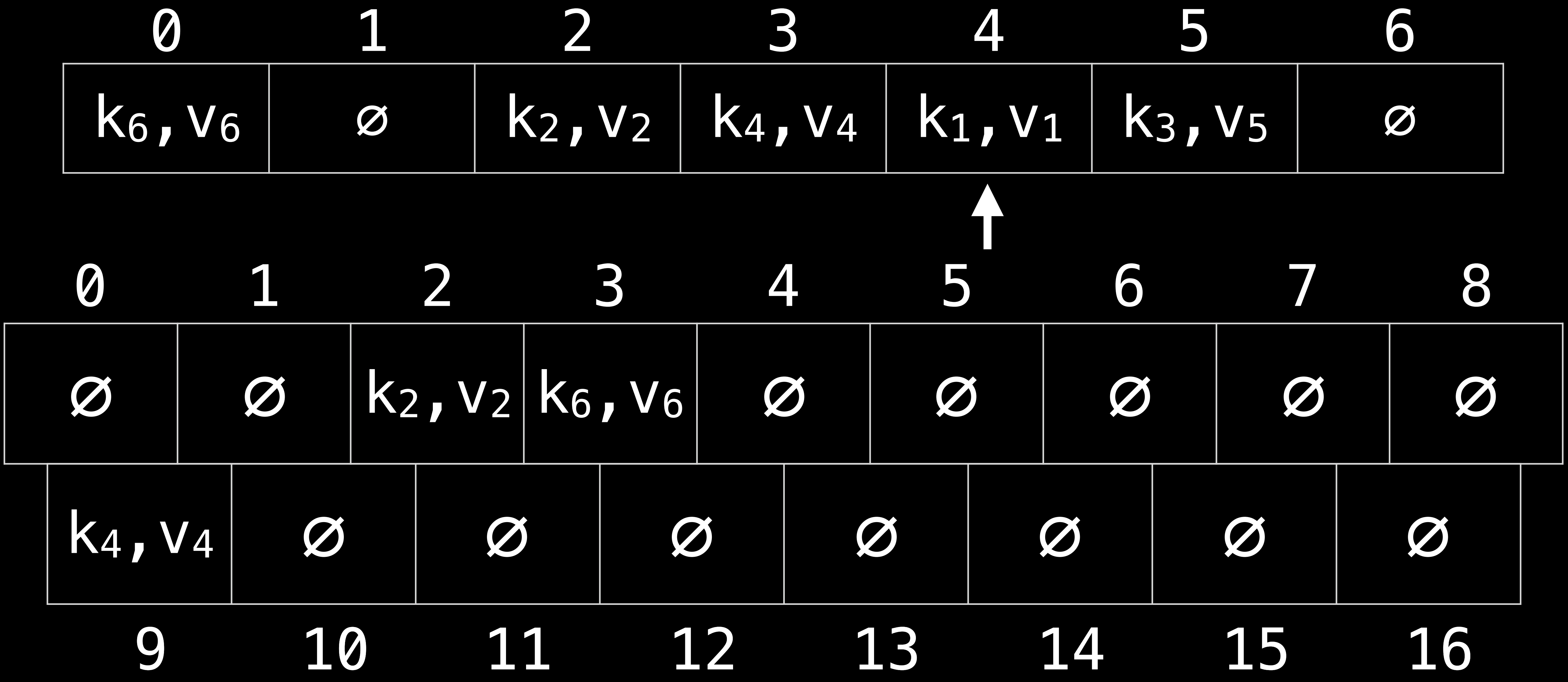
↑

0	1	2	3	4	5	6	7	8
∅	∅	k_2, v_2	k_6, v_6	∅	∅	∅	∅	∅

9	10	11	12	13	14	15	16
k_4, v_4	∅	∅	∅	∅	∅	∅	∅

From before, $\mathbf{H_1}(k_1) = 2$, $\mathbf{H_2}(k_1) = 34$

$$\delta = \mathbf{H_2}(k_1) \bmod 17 = 0$$



0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

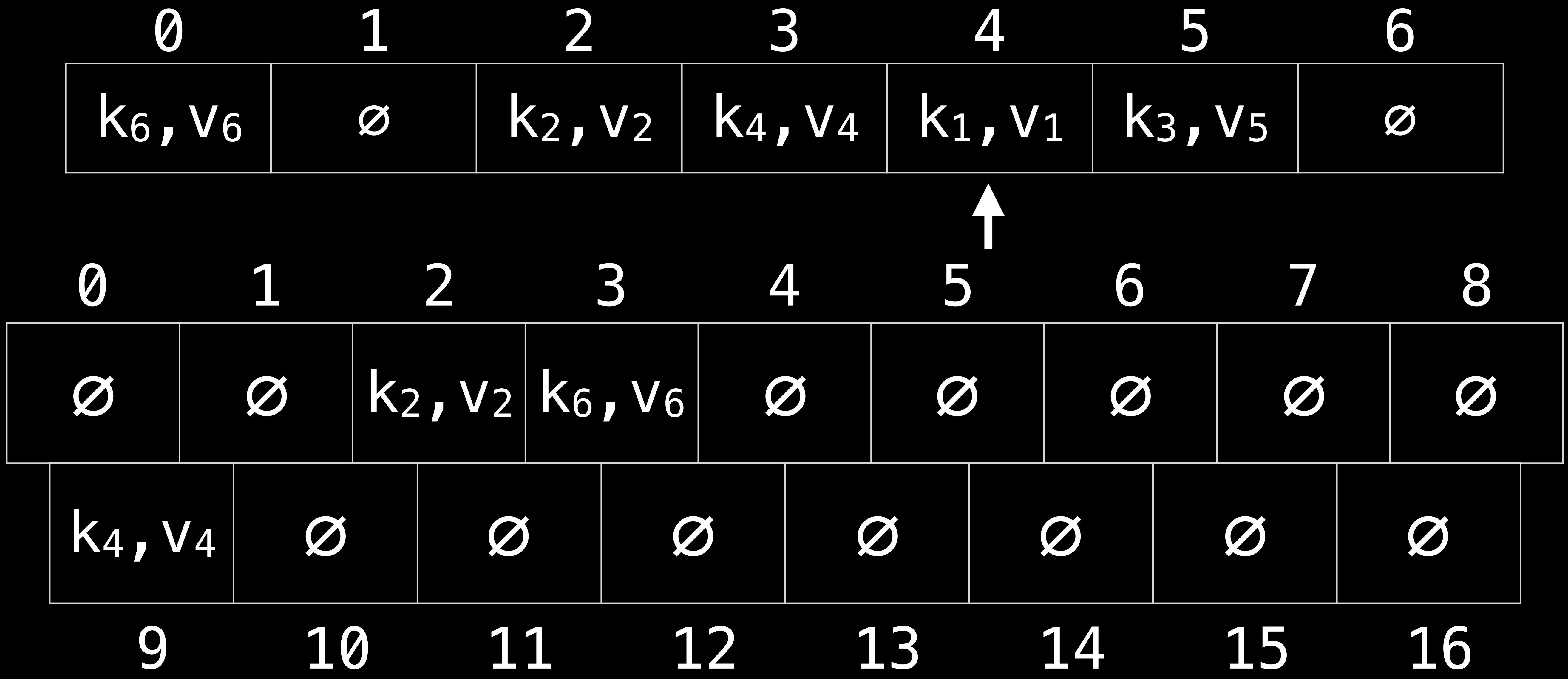
0	1	2	3	4	5	6	7	8
∅	∅	k_2, v_2	k_6, v_6	∅	∅	∅	∅	∅

k_4, v_4	∅	∅	∅	∅	∅	∅	∅
9	10	11	12	13	14	15	16

From before, $H_1(k_1) = 2$, $H_2(k_1) = 34$

$\delta = H_2(k_1) \bmod 17 = 0$

$\delta = 0$, so set $\delta = 1$



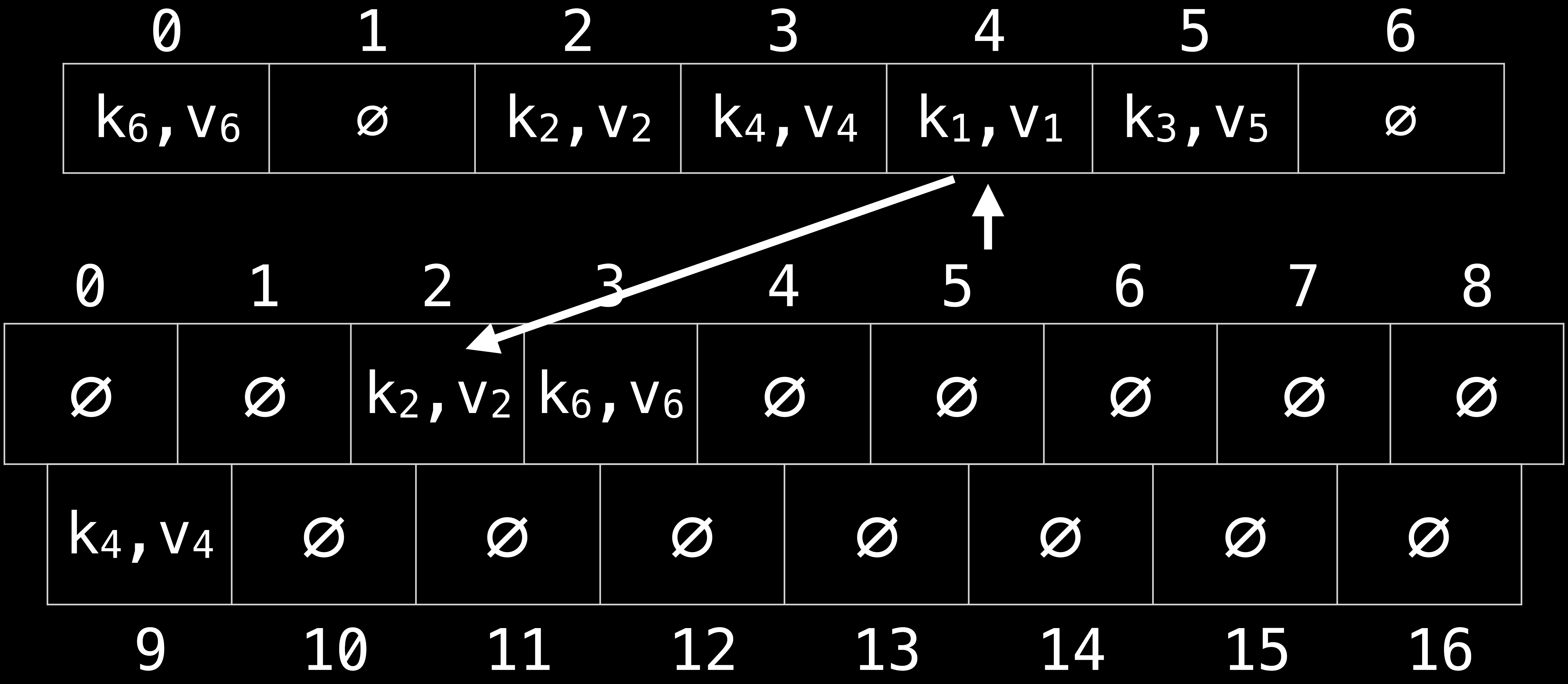
0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2, v_2	k_6, v_6	∅	∅	∅	∅	∅

9	10	11	12	13	14	15	16
k_4, v_4	∅	∅	∅	∅	∅	∅	∅

From before, **H_1**$(k_1) = 2$, **H_2**$(k_1) = 34$

$\delta = 1$



0	1	2	3	4	5	6
k_6,v_6	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2,v_2	k_6,v_6	∅	∅	∅	∅	∅

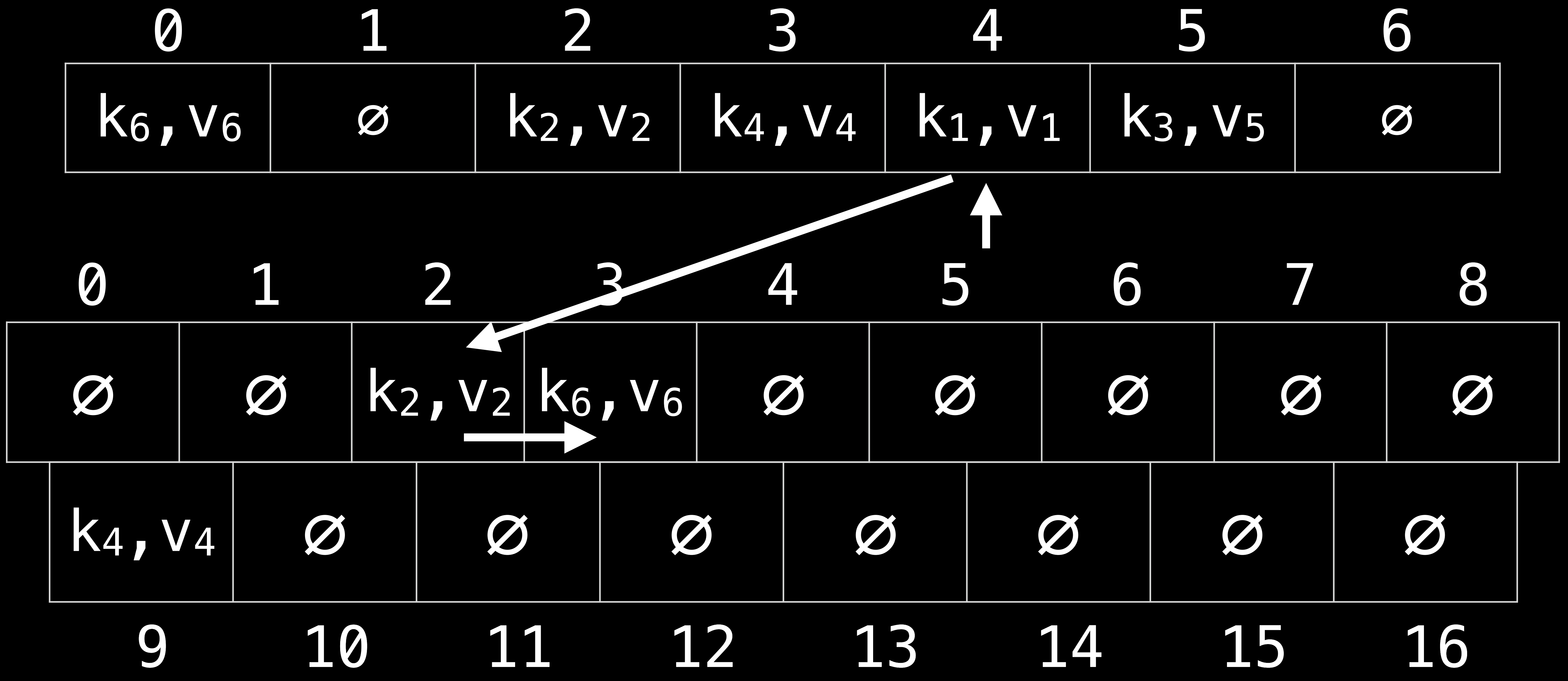
9	10	11	12	13	14	15	16
k_4,v_4	∅	∅	∅	∅	∅	∅	∅

From before, $H_1(k_1) = 2$, $H_2(k_1) = 34$

$$\delta = 1$$

$$H_1(k_1) + 0*\delta \bmod 17 = 2$$

Collision at bucket 2 in new table so keep probing



0	1	2	3	4	5	6
k_6,v_6	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2,v_2	k_6,v_6	∅	∅	∅	∅	∅

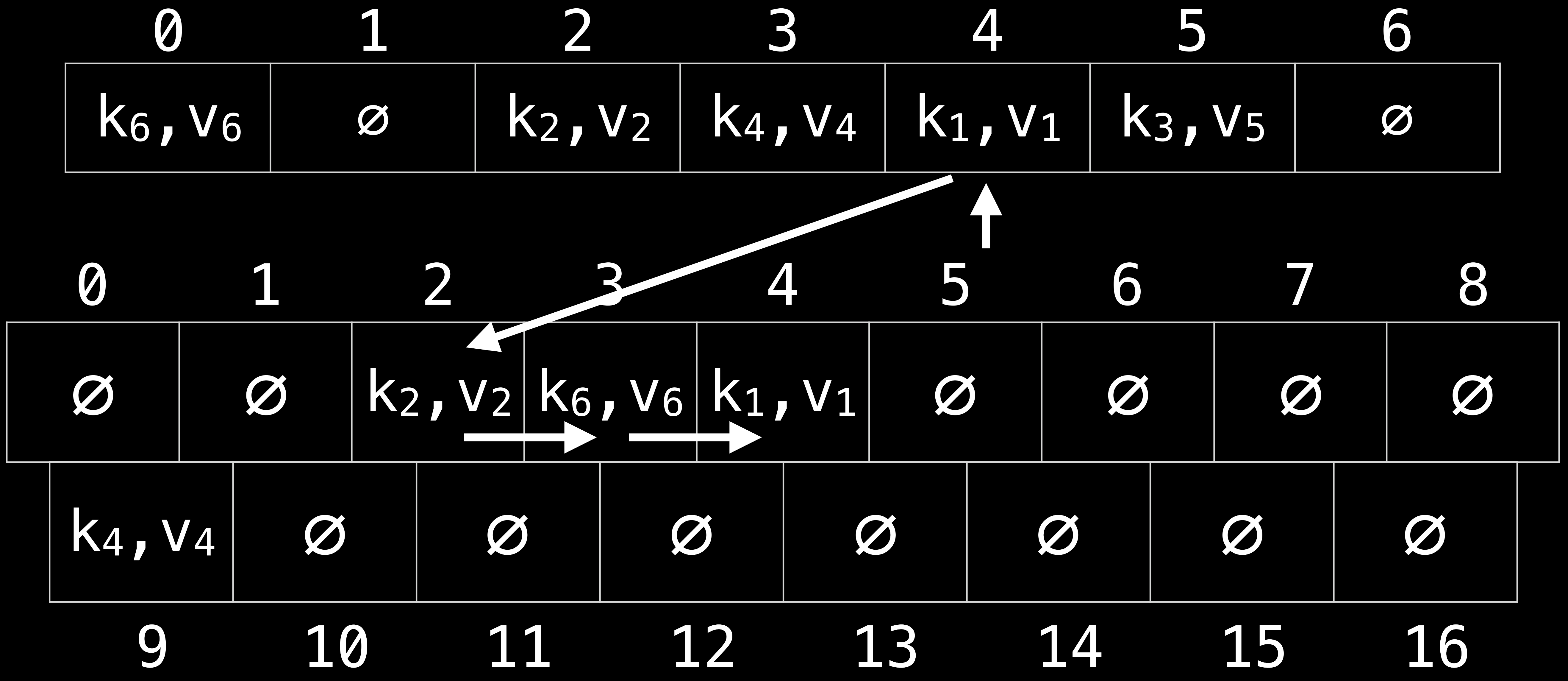
9	10	11	12	13	14	15	16
k_4,v_4	∅	∅	∅	∅	∅	∅	∅

From before, $\mathbf{H_1}(k_1) = 2$, $\mathbf{H_2}(k_1) = 34$

$\delta = 1$

$\mathbf{H_1}(k_1) + 0*\delta \bmod 17 = 2$
$\mathbf{H_1}(k_1) + 1*\delta \bmod 17 = 3$

Collision at bucket 3 in new
table so keep probing



0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

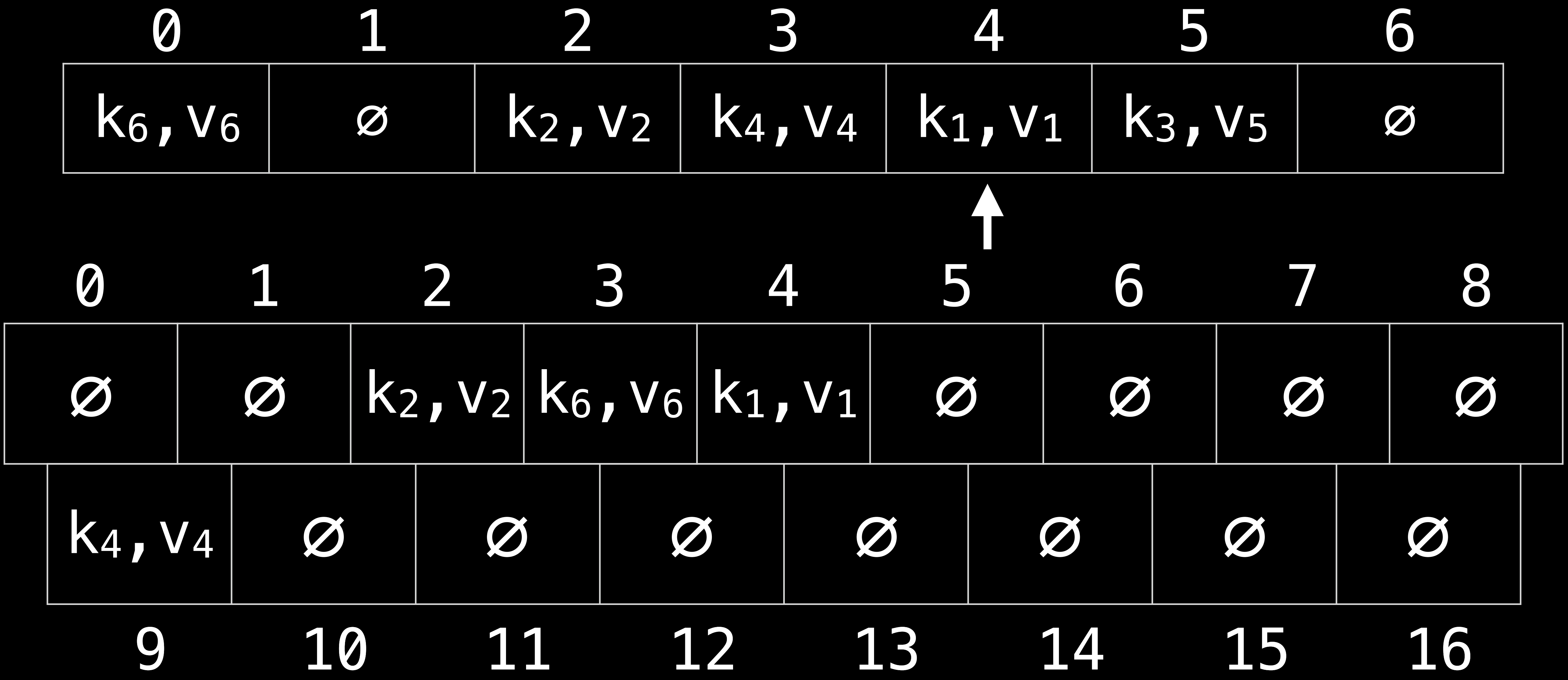
0	1	2	3	4	5	6	7	8
∅	∅	k_2, v_2	k_6, v_6	k_1, v_1	∅	∅	∅	∅

9	10	11	12	13	14	15	16
k_4, v_4	∅	∅	∅	∅	∅	∅	∅

From before, $H_1(k_1) = 2$, $H_2(k_1) = 34$

$$\delta = 1$$

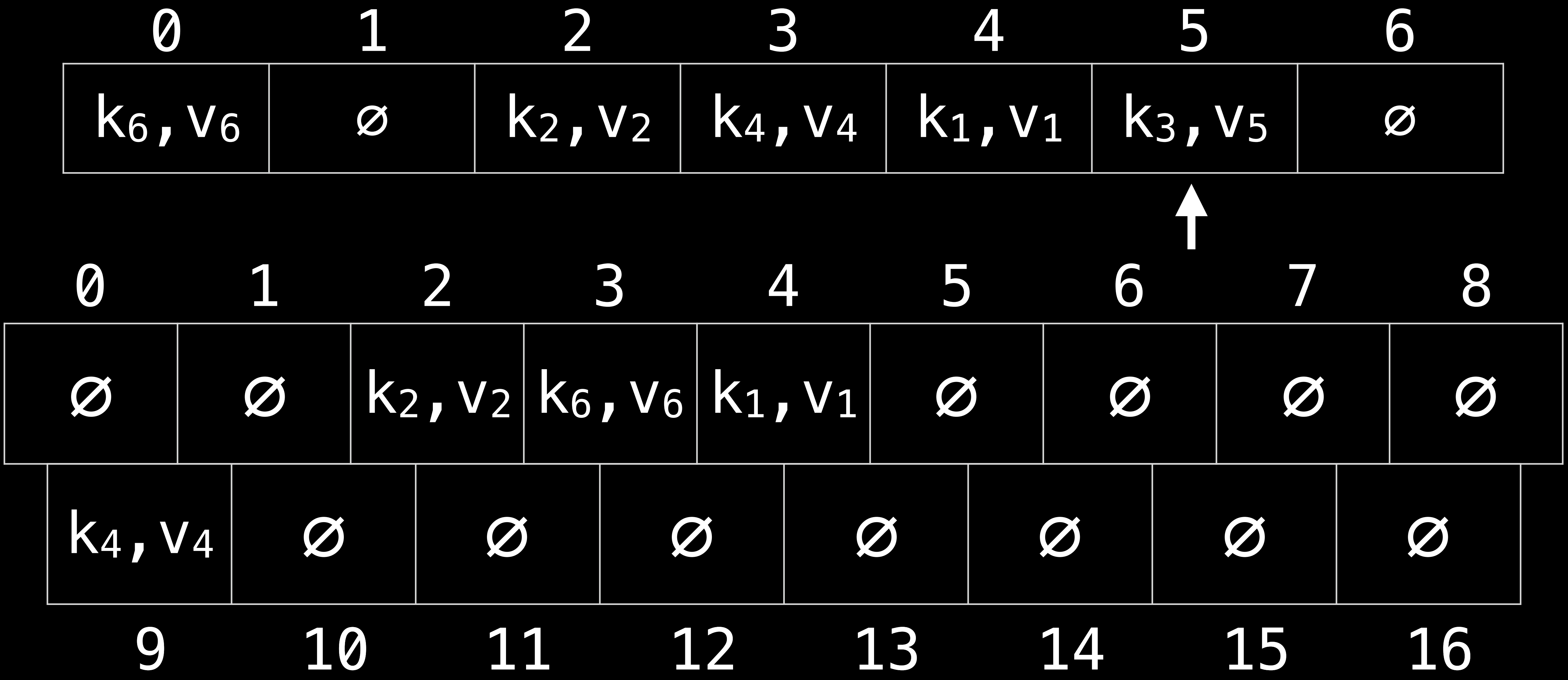
$$H_1(k_1) + 0*\delta \bmod 17 = 2$$
$$H_1(k_1) + 1*\delta \bmod 17 = 3$$
$$H_1(k_1) + 2*\delta \bmod 17 = 4$$



0	1	2	3	4	5	6
k_6,v_6	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2,v_2	k_6,v_6	k_1,v_1	∅	∅	∅	∅

9	10	11	12	13	14	15	16
k_4,v_4	∅	∅	∅	∅	∅	∅	∅



0	1	2	3	4	5	6
k_6,v_6	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2,v_2	k_6,v_6	k_1,v_1	∅	∅	∅	∅

k_4,v_4	∅	∅	∅	∅	∅	∅	∅
9	10	11	12	13	14	15	16

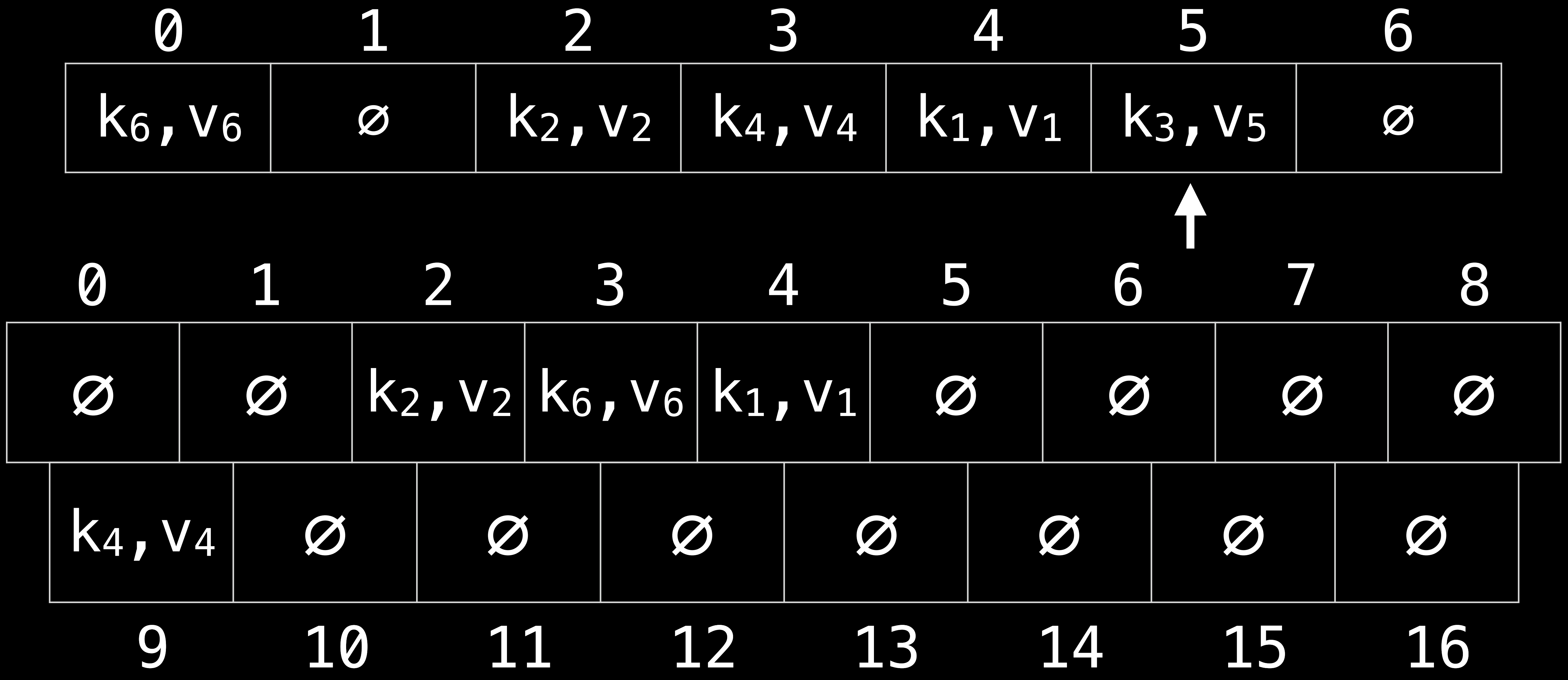
0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

↑

0	1	2	3	4	5	6	7	8
∅	∅	k_2, v_2	k_6, v_6	k_1, v_1	∅	∅	∅	∅

k_4, v_4	∅	∅	∅	∅	∅	∅	∅
9	10	11	12	13	14	15	16

From before, $\mathbf{H_1}(k_3) = 2$, $\mathbf{H_2}(k_3) = 10$



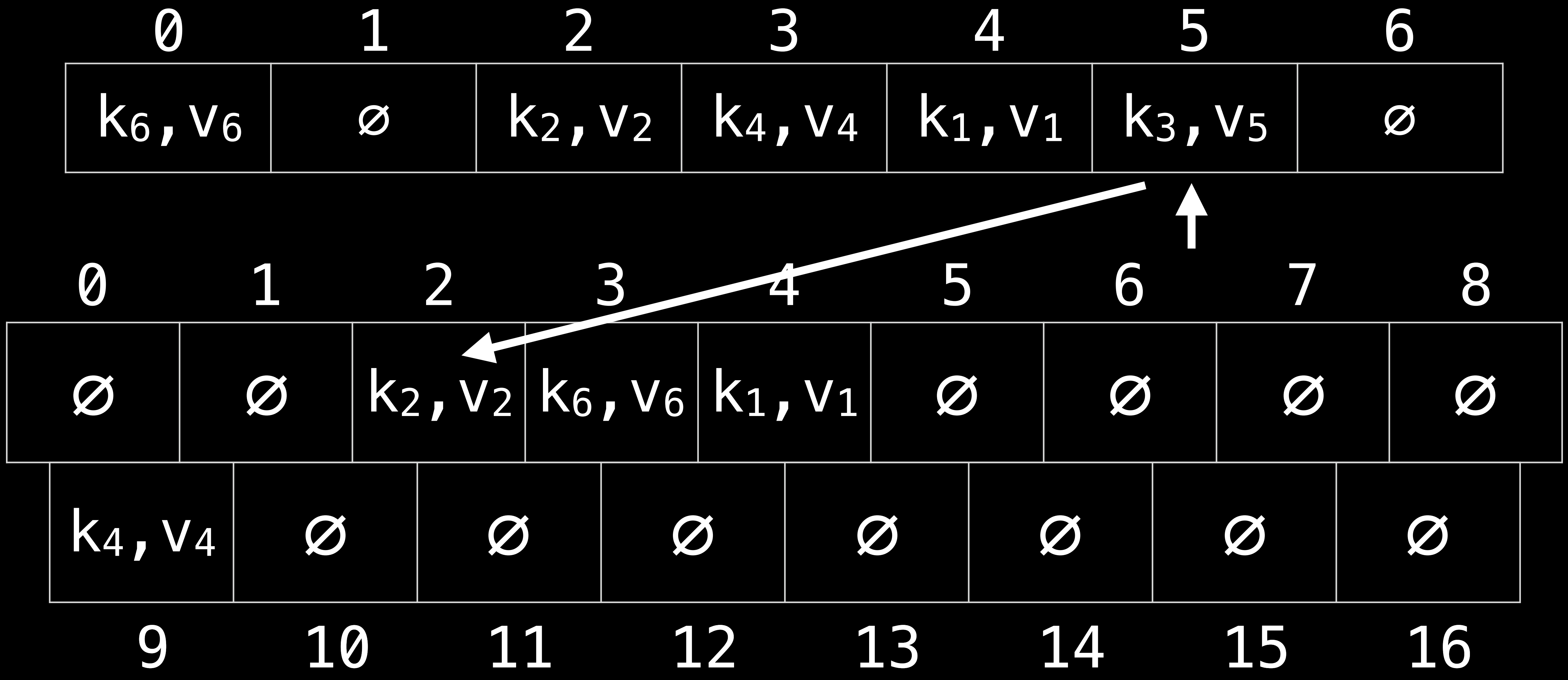
0	1	2	3	4	5	6
k_6, v_6	∅	k_2, v_2	k_4, v_4	k_1, v_1	k_3, v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2, v_2	k_6, v_6	k_1, v_1	∅	∅	∅	∅

9	10	11	12	13	14	15	16
k_4, v_4	∅	∅	∅	∅	∅	∅	∅

From before, $\mathbf{H_1}(k_3) = 2$, $\mathbf{H_2}(k_3) = 10$

$\delta = \mathbf{H_2}(k_3) \bmod 17 = 10$



0	1	2	3	4	5	6
k_6,v_6	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2,v_2	k_6,v_6	k_1,v_1	∅	∅	∅	∅

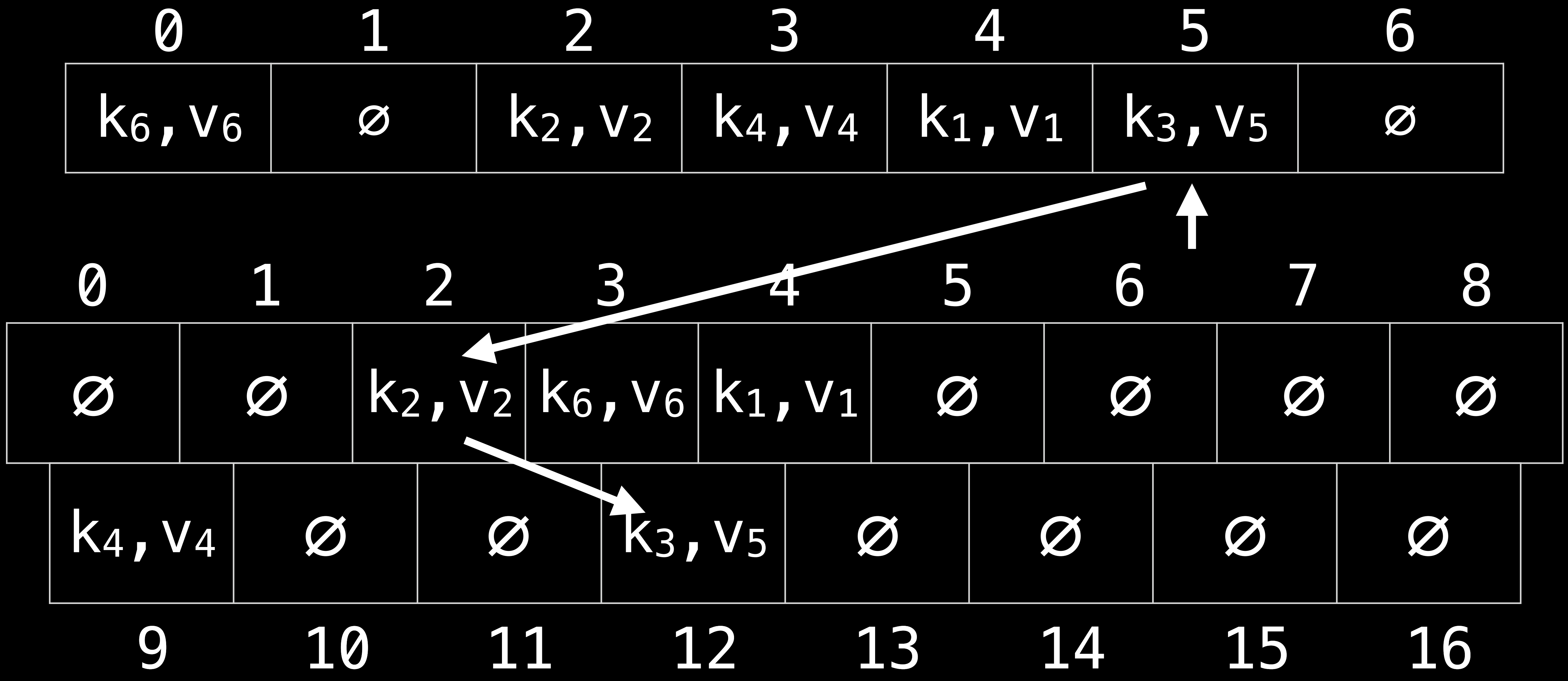
9	10	11	12	13	14	15	16
k_4,v_4	∅	∅	∅	∅	∅	∅	∅

From before, $\mathbf{H_1}(k_3) = 2$, $\mathbf{H_2}(k_3) = 10$

$$\delta = \mathbf{H_2}(k_3) \bmod 17 = 10$$

$$\mathbf{H_1}(k_3) + 0*\delta \bmod 17 = 2$$

Collision again at bucket 2 so keep probing!



0	1	2	3	4	5	6
k_6,v_6	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2,v_2	k_6,v_6	k_1,v_1	∅	∅	∅	∅

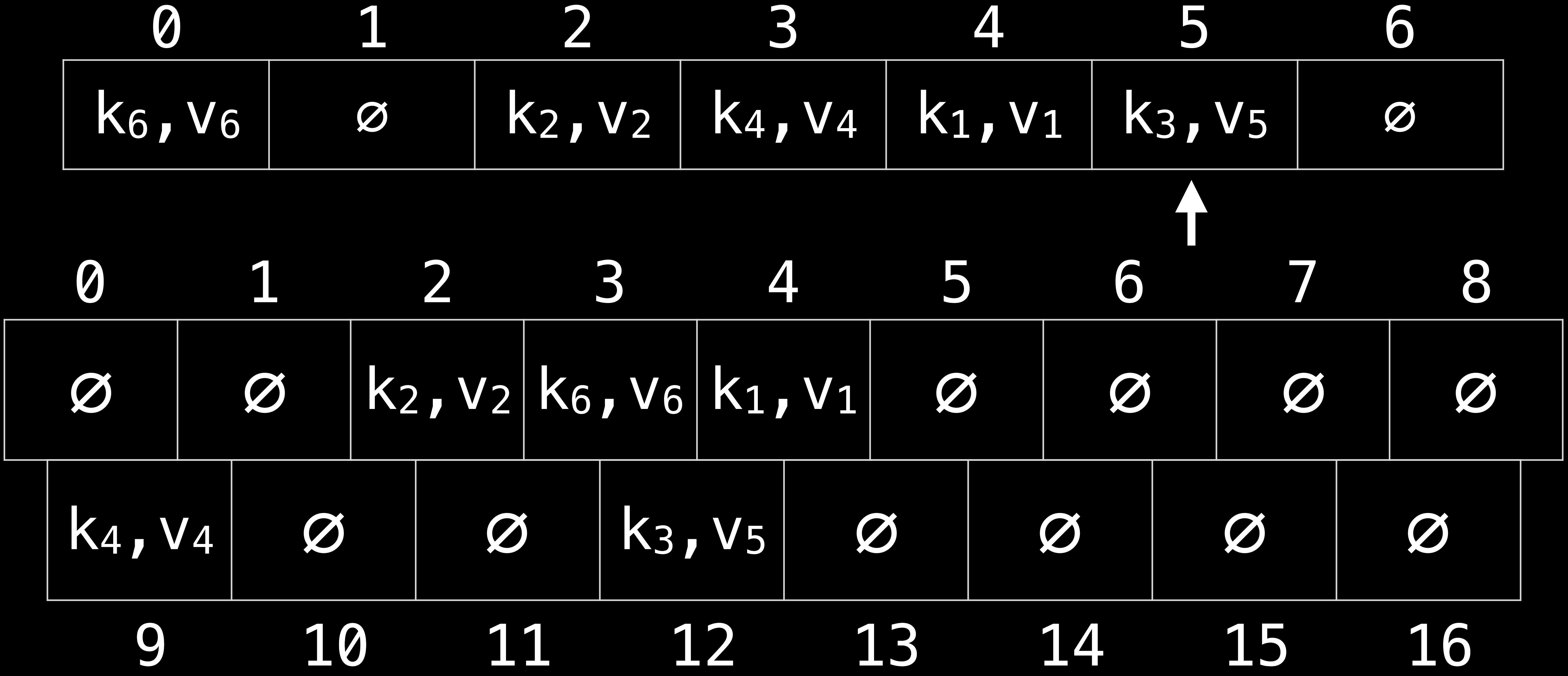
9	10	11	12	13	14	15	16
k_4,v_4	∅	∅	k_3,v_5	∅	∅	∅	∅

From before, $\mathbf{H_1}(k_3) = 2$, $\mathbf{H_2}(k_3) = 10$

$$\delta = \mathbf{H_2}(k_3) \bmod 17 = 10$$

$$\mathbf{H_1}(k_3) + 0 * \delta \bmod 17 = 2$$

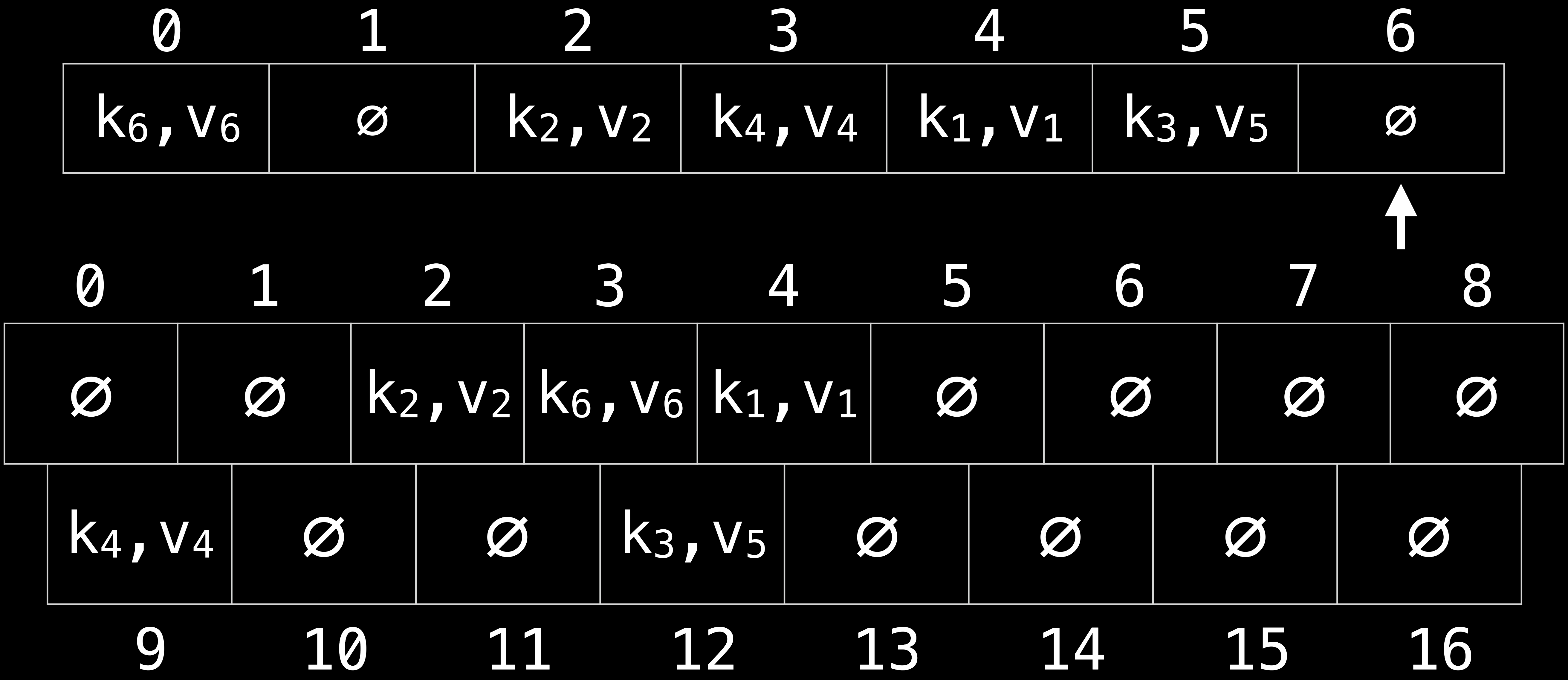
$$\mathbf{H_1}(k_3) + 1 * \delta \bmod 17 = 12$$



0	1	2	3	4	5	6
k_6,v_6	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2,v_2	k_6,v_6	k_1,v_1	∅	∅	∅	∅

9	10	11	12	13	14	15	16
k_4,v_4	∅	∅	k_3,v_5	∅	∅	∅	∅



0	1	2	3	4	5	6
k_6,v_6	∅	k_2,v_2	k_4,v_4	k_1,v_1	k_3,v_5	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2,v_2	k_6,v_6	k_1,v_1	∅	∅	∅	∅

9	10	11	12	13	14	15	16
k_4,v_4	∅	∅	k_3,v_5	∅	∅	∅	∅

0	1	2	3	4	5	6	7	8
∅	∅	k_2, v_2	k_6, v_6	k_1, v_1	∅	∅	∅	∅

9	10	11	12	13	14	15	16
k_4, v_4	∅	∅	k_3, v_5	∅	∅	∅	∅

Operations: Suppose $H_1(k_7) = 15$, $H_2(k_7) = 3$

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

0	1	2	3	4	5	6	7	8
∅	∅	k_2,v_2	k_6,v_6	k_1,v_1	∅	∅	∅	∅

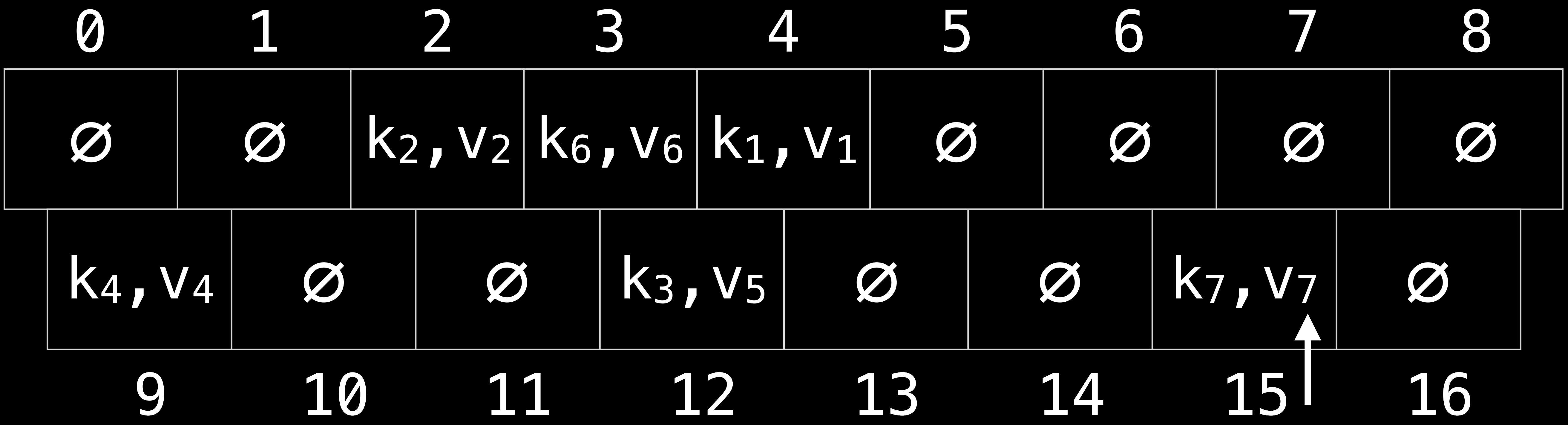
9	10	11	12	13	14	15	16
k_4,v_4	∅	∅	k_3,v_5	∅	∅	∅	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose **H_1**(k_7) = 15, **H_2**(k_7) = 3

$\delta = H_2(k_7) \bmod 17 = 3$



0	1	2	3	4	5	6	7	8
∅	∅	k_2,v_2	k_6,v_6	k_1,v_1	∅	∅	∅	∅

9	10	11	12	13	14	15	16
k_4,v_4	∅	∅	k_3,v_5	∅	∅	k_7,v_7	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Suppose $H_1(k_7) = 15$, $H_2(k_7) = 3$

$$\delta = H_2(k_7) \bmod 17 = 3$$

$$H_1(k_7) + 0*\delta \bmod 17 = 15$$

0	1	2	3	4	5	6	7	8
∅	∅	k_2,v_2	k_6,v_6	k_1,v_1	∅	∅	∅	∅

9	10	11	12	13	14	15	16
k_4,v_4	∅	∅	k_3,v_5	∅	∅	k_7,v_7	∅

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
insert(k_4,v_4)
insert(k_3,v_5)
insert(k_6,v_6)
insert(k_7,v_7)

Next Video:
Removing from a hash table

Double hashing implementation source code and tests can all be found at:

github.com/williamfiset/data-structures

Hash table (HT) Removing elements open addressing

A quick guide to removing key-value pairs
in a hash table via open addressing

William Fiset

Issues with removing

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	∅	∅

Suppose we have an empty hash table and we’re using linear probing with **P**(x) = x as our probing function.

Issues with removing

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	∅	∅

Recall that **P**(x) = x, N = 8

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

Assume for the sake of argument that **H**(k_1) = **H**(k_2) = **H**(k_3) = 1

Issues with removing

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	∅	∅

Recall that $P(x) = x$, $N = 8$

$H(k_1) = 1$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

Issues with removing

0	1	2	3	4	5	6	7
∅	∅	∅	∅	∅	∅	∅	∅

Recall that $P(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

$$H(k_1) = 1$$

$$H(k_1) + P(0) \bmod N = 1$$

$$1 + 0 \bmod 8 = 1$$

Issues with removing

0	1	2	3	4	5	6	7
∅	k_1,v_1	∅	∅	∅	∅	∅	∅

↑

Recall that $P(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

$$H(k_1) = 1$$

$$H(k_1) + P(0) \bmod N = 1$$

$$1 + 0 \bmod 8 = 1$$

Issues with removing

0	1	2	3	4	5	6	7
∅	k_1,v_1	∅	∅	∅	∅	∅	∅

Recall that $P(x) = x$, $N = 8$

Operations:

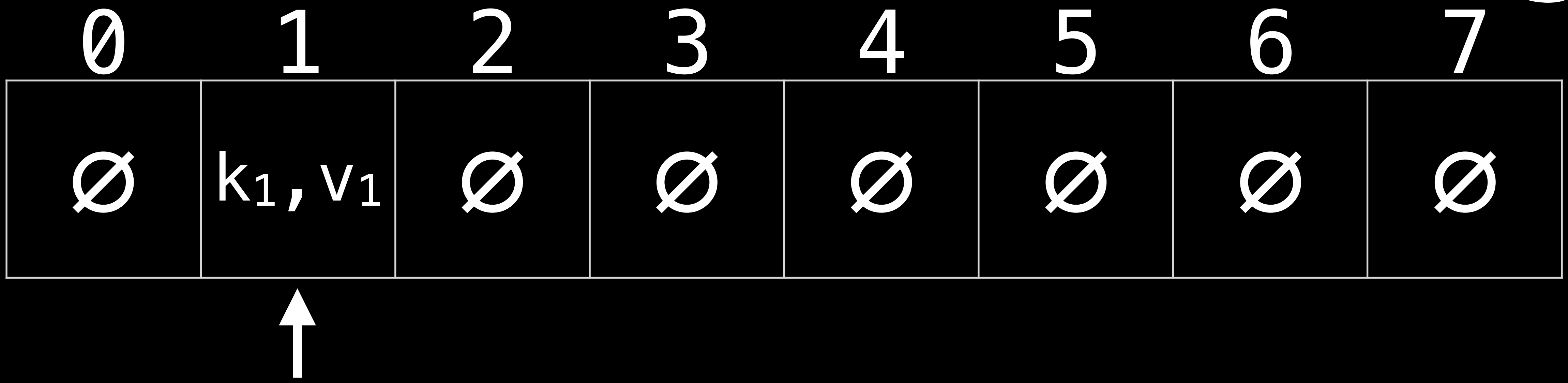
insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

$$H(k_2) = 1$$

$$H(k_2) + P(0) \bmod N = 1$$

$$1 + 0 \bmod 8 = 1$$

Issues with removing



0	1	2	3	4	5	6	7
∅	k_1,v_1	∅	∅	∅	∅	∅	∅

Recall that $P(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

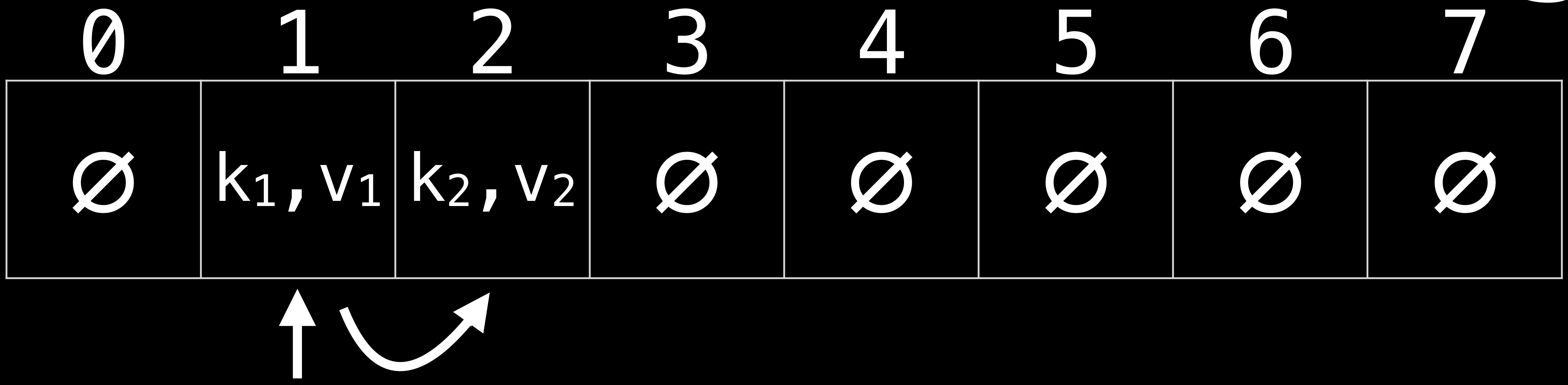
$$H(k_2) = 1$$

$$H(k_2) + P(0) \bmod N = 1$$

$$1 + 0 \bmod 8 = 1$$

Bucket 1 is occupied, so keep probing.

Issues with removing



0	1	2	3	4	5	6	7
∅	k_1,v_1	k_2,v_2	∅	∅	∅	∅	∅

Recall that $P(x) = x$, $N = 8$

Operations:

$\text{insert}(k_1,v_1)$

$\text{insert}(k_2,v_2)$

$\text{insert}(k_3,v_3)$

$\text{remove}(k_2)$

$\text{getValue}(k_3)$

$$H(k_2) = 1$$

$$H(k_2) + P(0) \bmod N = 1$$

$$1 + 0 \bmod 8 = 1$$

$$H(k_2) + P(1) \bmod N = 2$$

$$1 + 1 \bmod 8 = 2$$

Issues with removing

0	1	2	3	4	5	6	7
∅	k_1,v_1	k_2,v_2	∅	∅	∅	∅	∅

Recall that $P(x) = x, N = 8$

<u>Operations</u>:

$insert(k_1,v_1)$
$insert(k_2,v_2)$
$insert(k_3,v_3)$
$remove(k_2)$
$getValue(k_3)$

$H(k_3) = 1$

Issues with removing

0	1	2	3	4	5	6	7
∅	k_1,v_1	k_2,v_2	∅	∅	∅	∅	∅

Recall that $P(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

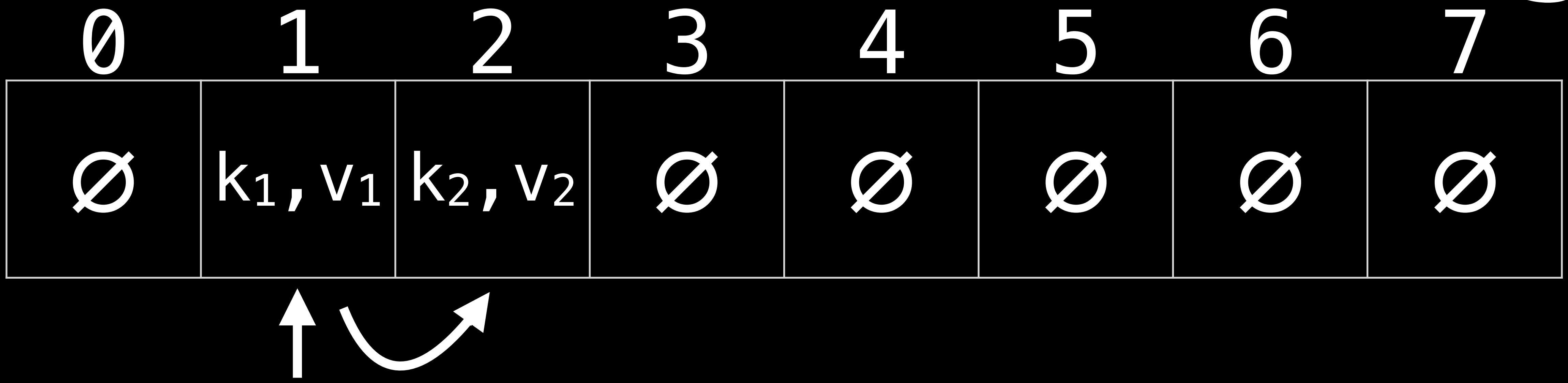
$$H(k_3) = 1$$

$$H(k_3) + P(0) \bmod N = 1$$

$$1 + 0 \bmod 8 = 1$$

Bucket 1 is occupied, so keep probing.

Issues with removing



0	1	2	3	4	5	6	7
∅	k_1, v_1	k_2, v_2	∅	∅	∅	∅	∅

Recall that $P(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

$$H(k_3) = 1$$

$$H(k_3) + P(0) \bmod N = 1$$

$$1 + 0 \bmod 8 = 1$$

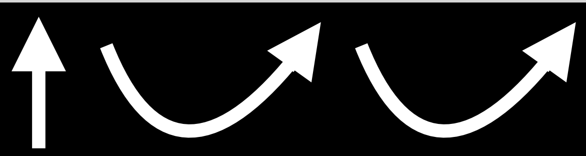
$$H(k_3) + P(1) \bmod N = 2$$

$$1 + 1 \bmod 8 = 2$$

Bucket 2 is occupied, so keep probing.

Issues with removing

0	1	2	3	4	5	6	7
∅	k_1, v_1	k_2, v_2	k_3, v_3	∅	∅	∅	∅



Recall that $P(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)

insert(k_2,v_2)

insert(k_3,v_3)

remove(k_2)

getValue(k_3)

$$H(k_3) = 1$$

$$H(k_3) + P(0) \bmod N = 1$$

$$1 + 0 \bmod 8 = 1$$

$$H(k_3) + P(1) \bmod N = 2$$

$$1 + 1 \bmod 8 = 2$$

$$H(k_3) + P(2) \bmod N = 3$$

$$1 + 2 \bmod 8 = 3$$

Issues with removing

0	1	2	3	4	5	6	7
∅	k_1, v_1	k_2, v_2	k_3, v_3	∅	∅	∅	∅

Recall that **P**$(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

For this example we’ll use naive removing where we just clear the bucket and explore why that doesn’t quite work.

Issues with removing

0	1	2	3	4	5	6	7
∅	k_1, v_1	k_2, v_2	k_3, v_3	∅	∅	∅	∅

Recall that **P**$(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

H$(k_2) = 1$

Issues with removing

0	1	2	3	4	5	6	7
∅	k_1, v_1	k_2, v_2	k_3, v_3	∅	∅	∅	∅

Recall that $P(x) = x$, $N = 8$

Operations:

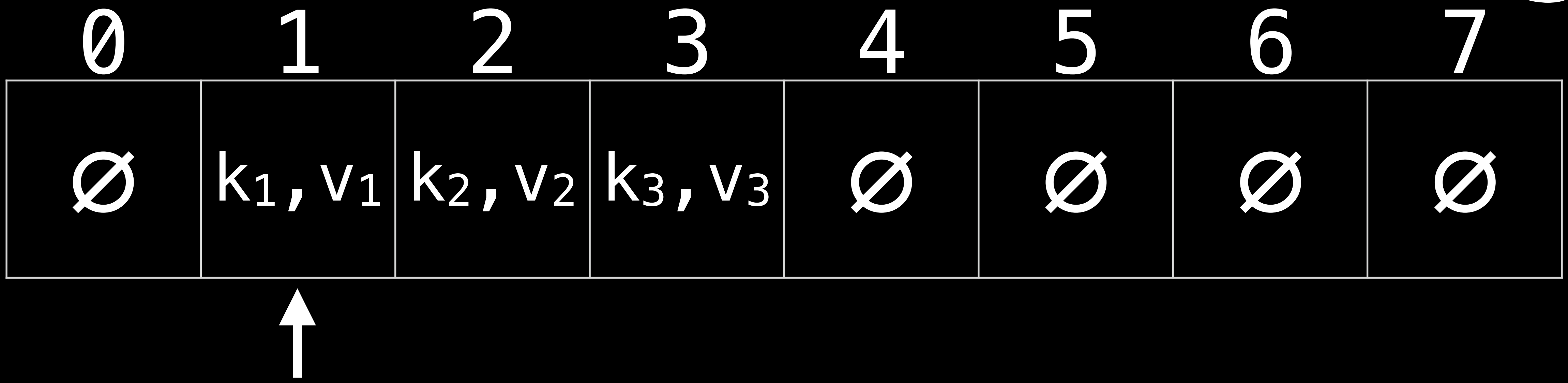
insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

$$H(k_2) = 1$$

$$H(k_2) + P(0) \bmod N = 1$$

$$1 + 0 \bmod 8 = 1$$

Issues with removing



0	1	2	3	4	5	6	7
∅	k_1,v_1	k_2,v_2	k_3,v_3	∅	∅	∅	∅

Recall that $P(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)

insert(k_2,v_2)

insert(k_3,v_3)

remove(k_2)

getValue(k_3)

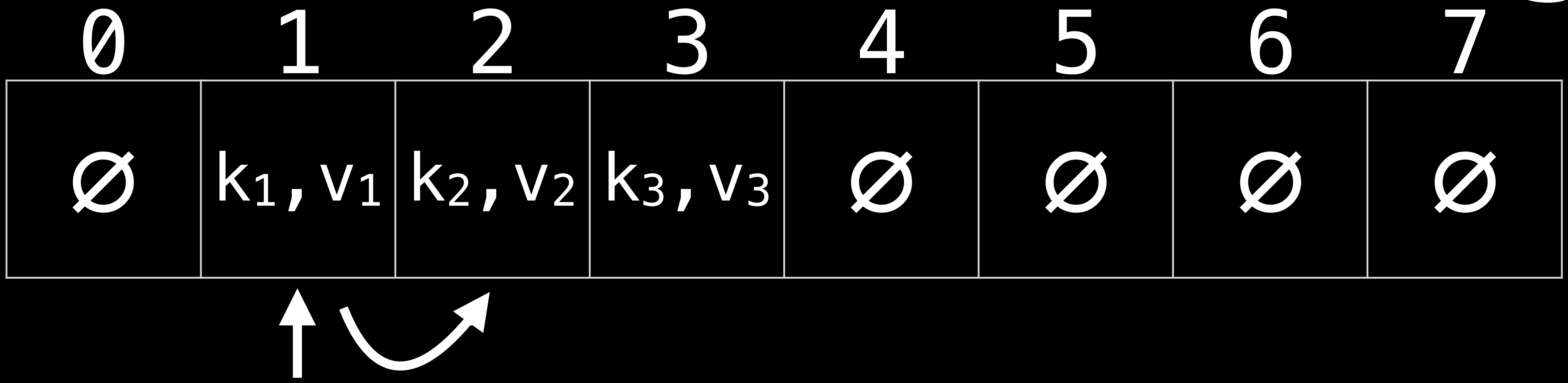
$$H(k_2) = 1$$

$$H(k_2) + P(0) \bmod N = 1$$

$$1 + 0 \bmod 8 = 1$$

Look in bucket at index 1 and discover that k_1 is not equal to k_2 so the search continues…

Issues with removing



0	1	2	3	4	5	6	7
∅	k_1,v_1	k_2,v_2	k_3,v_3	∅	∅	∅	∅

Recall that $P(x) = x$, $N = 8$

Operations:

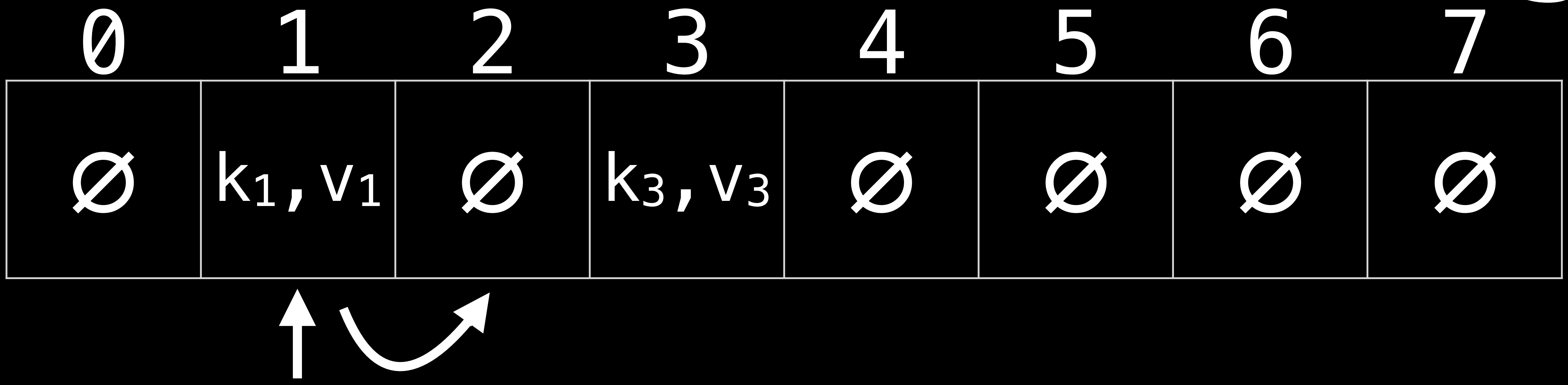
insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

$$H(k_2) = 1$$

$$H(k_2) + P(0) \bmod N = 1$$
$$1 + 0 \bmod 8 = 1$$
$$H(k_2) + P(1) \bmod N = 2$$
$$1 + 1 \bmod 8 = 2$$

In bucket at index 2 the key k_2 is found!

Issues with removing



0	1	2	3	4	5	6	7
∅	k_1,v_1	∅	k_3,v_3	∅	∅	∅	∅

Recall that $P(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

$$H(k_2) = 1$$

$$H(k_2) + P(0) \bmod N = 1$$

$$1 + 0 \bmod 8 = 1$$

$$H(k_2) + P(1) \bmod N = 2$$

$$1 + 1 \bmod 8 = 2$$

In bucket at index 2 the key k_2 is found!

Issues with removing

0	1	2	3	4	5	6	7
∅	k_1,v_1	∅	k_3,v_3	∅	∅	∅	∅

Recall that **P**(x) = x, N = 8

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

Now let's query the value of k_3 inside our hashtable.

Issues with removing

0	1	2	3	4	5	6	7
∅	k_1,v_1	∅	k_3,v_3	∅	∅	∅	∅

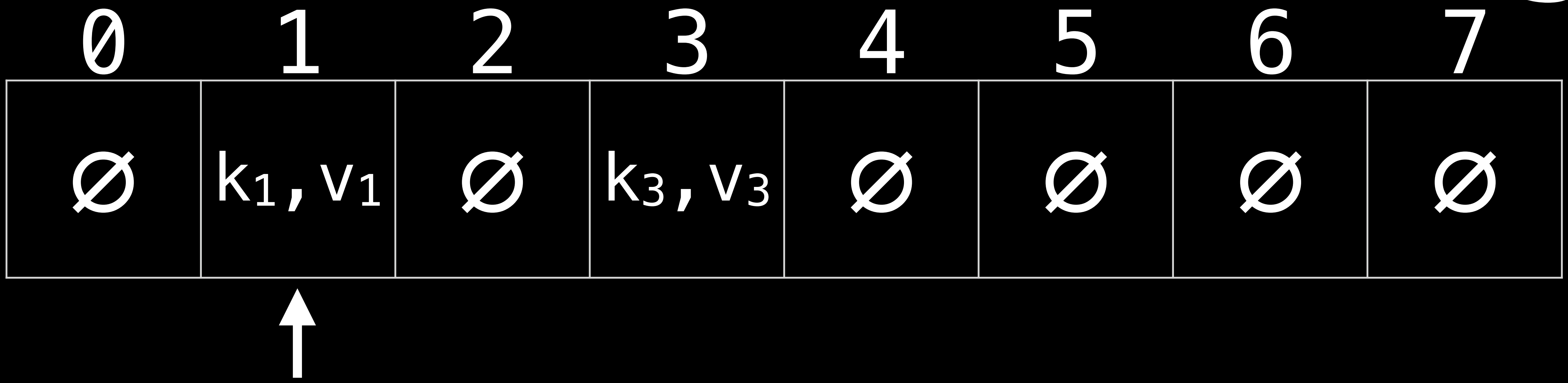
Recall that $\mathbf{P}(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

$\mathbf{H}(k_3) = 1$

Issues with removing



0	1	2	3	4	5	6	7
∅	k_1,v_1	∅	k_3,v_3	∅	∅	∅	∅

Recall that $P(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

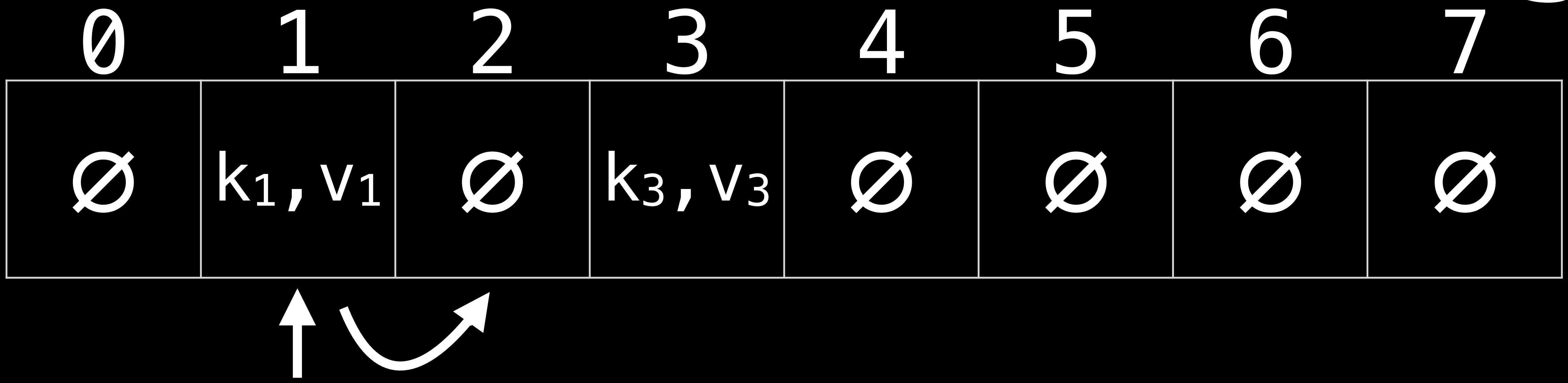
$$H(k_3) = 1$$

$$H(k_3) + P(0) \bmod N = 1$$

$$1 + 0 \bmod 8 = 1$$

In bucket 1 $k_1 \neq k_3$ so continue the search.

Issues with removing



0	1	2	3	4	5	6	7
∅	k_1,v_1	∅	k_3,v_3	∅	∅	∅	∅

Recall that $P(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)

insert(k_2,v_2)

insert(k_3,v_3)

remove(k_2)

getValue(k_3)

$$H(k_3) = 1$$

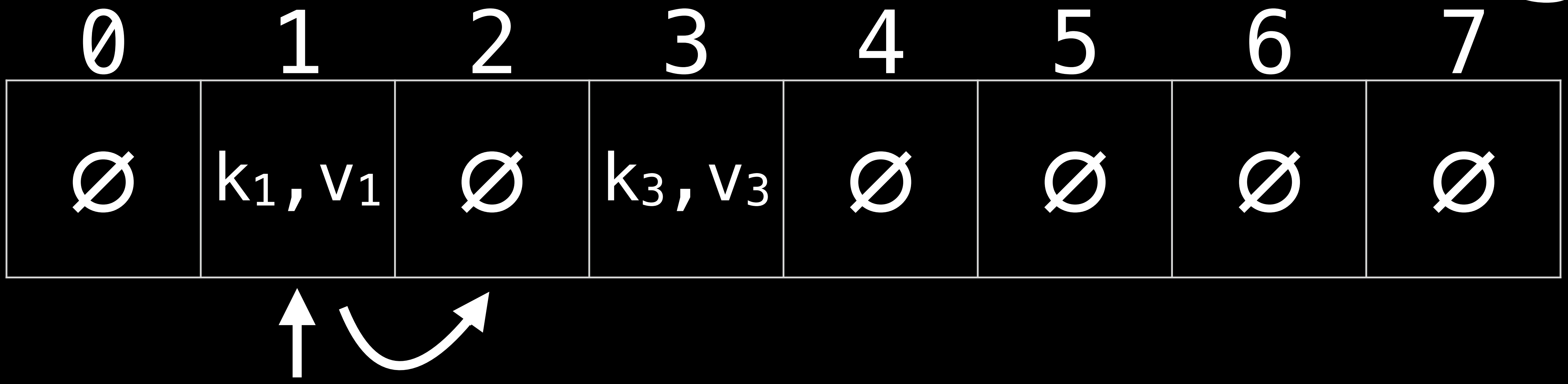
$$H(k_3) + P(0) \bmod N = 1$$

$$1 + 0 \bmod 8 = 1$$

$$H(k_3) + P(1) \bmod N = 2$$

$$1 + 1 \bmod 8 = 2$$

Issues with removing



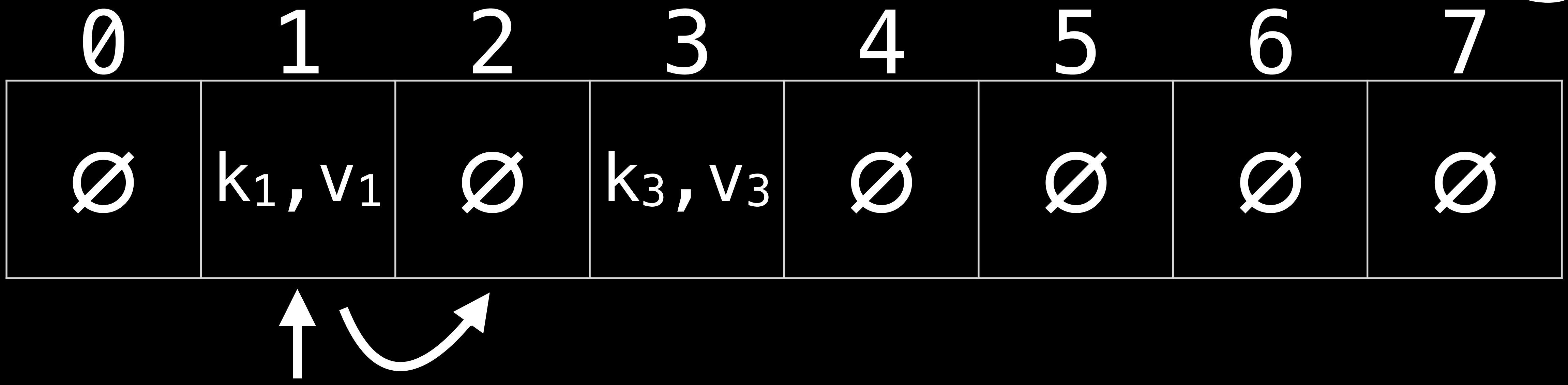
0	1	2	3	4	5	6	7
∅	k_1,v_1	∅	k_3,v_3	∅	∅	∅	∅

Recall that **P**(x) = x, N = 8

Operations:
insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

The value in the bucket at index 2 is null so we must conclude that the key k_3 does not exist in the hash table otherwise we would have found it before reaching a null position!

Issues with removing



0	1	2	3	4	5	6	7
∅	k_1, v_1	∅	k_3, v_3	∅	∅	∅	∅

Recall that $\mathbf{P}(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

However, the key k_3 clearly exists in our table! Hence, the naive removing method doesn’t work :/

Solution to removing

0	1	2	3	4	5	6	7
∅	k_1,v_1	∅	k_3,v_3	∅	∅	∅	∅

The solution is to place a **unique marker** called a **tombstone** instead of null to indicate that a (k,v) pair has been deleted and that the bucket should be skipped during a search.

Solution to removing

0	1	2	3	4	5	6	7
∅	k_1,v_1		k_3,v_3	∅	∅	∅	∅

Let’s replace the deleted bucket with a tombstone as we should have done and see what should have happened when we searched for k_3.

Solution to removing

0	1	2	3	4	5	6	7
∅	k_1, v_1	RIP	k_3, v_3	∅	∅	∅	∅

Recall that P(x) = x, N = 8

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

Solution to removing

0	1	2	3	4	5	6	7
∅	k_1,v_1	RIP	k_3,v_3	∅	∅	∅	∅

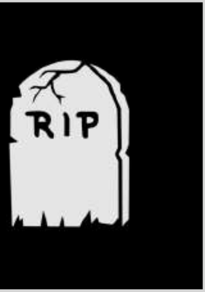
Recall that $P(x) = x$, $N = 8$

$H(k_3) = 1$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

Solution to removing

0	1	2	3	4	5	6	7
∅	k_1, v_1		k_3, v_3	∅	∅	∅	∅

↑

Recall that $P(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

$$H(k_3) = 1$$

$$H(k_3) + P(0) \bmod N = 1$$
$$1 + 0 \bmod 8 = 1$$

$k_1 \neq k_3$, so keep probing

Solution to removing

0	1	2	3	4	5	6	7
∅	k_1, v_1		k_3, v_3	∅	∅	∅	∅

Recall that $P(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)

insert(k_2,v_2)

insert(k_3,v_3)

remove(k_2)

getValue(k_3)

$$H(k_3) = 1$$

$$H(k_3) + P(0) \bmod N = 1$$

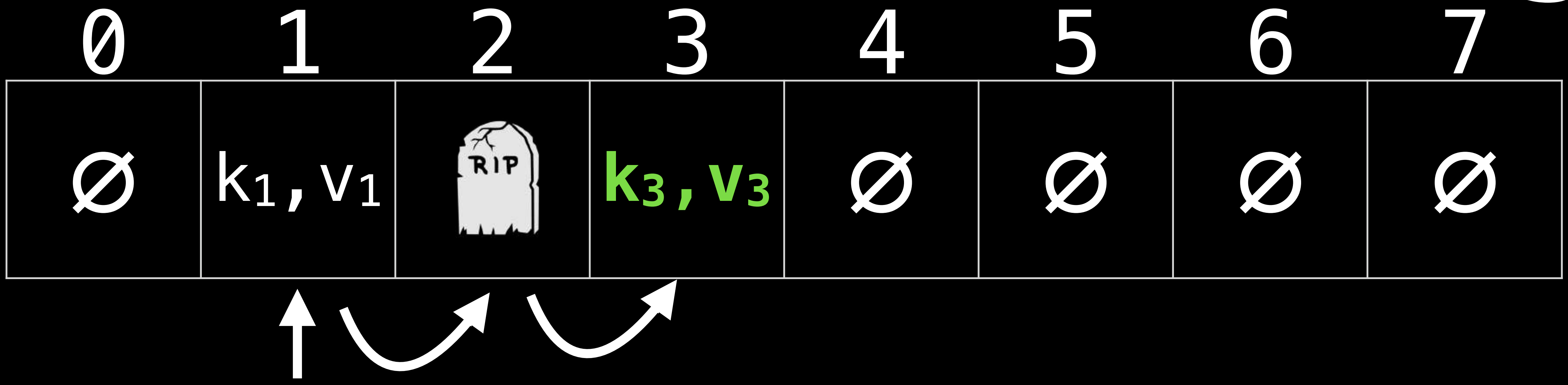
$$1 + 0 \bmod 8 = 1$$

$$H(k_3) + P(1) \bmod N = 2$$

$$1 + 1 \bmod 8 = 2$$

Hit a tombstone, so keep searching.

Solution to removing



0	1	2	3	4	5	6	7
∅	k_1, v_1	RIP	k_3, v_3	∅	∅	∅	∅

Recall that $P(x) = x$, $N = 8$

Operations:

insert(k_1,v_1)
insert(k_2,v_2)
insert(k_3,v_3)
remove(k_2)
getValue(k_3)

$H(k_3) = 1$

$H(k_3) + P(0) \bmod N = 1$

$1 + 0 \bmod 8 = 1$

$H(k_3) + P(1) \bmod N = 2$

$1 + 1 \bmod 8 = 2$

$H(k_3) + P(2) \bmod N = 3$

$1 + 2 \bmod 8 = 3$

Found k_3! Return v_3 as answer.

Tombstone question

Q: I have a lot of tombstones cluttering my HT how do I get rid of them?

A: Tombstones count as filled slots in the HT so they increase the load factor and will be removed when the table is resized. Additionally, when inserting a new (k,v) pair you can replace buckets with tombstones with the new key-value pair.

Inserting with 🪦s

0	1	2	3	4	5	6	7
🪦	∅	∅	🪦	∅	k_4,v_4	🪦	k_7,v_7

Suppose we have the following HT with the quadratic probing function **P**(x) = $(x^2+x)/2$. Let’s see how to delete tombstones while doing a lookup.

Inserting with 🪦s

0	1	2	3	4	5	6	7
🪦	∅	∅	🪦	∅	k_4, v_4	🪦	k_7, v_7

Recall that $P(x) = (x^2+x)/2$

Suppose we want to find the value of k_7 inside the HT and $H(k_7) = 5$.

Inserting with 🪦s

0	1	2	3	4	5	6	7
🪦	∅	∅	🪦	∅	k_4,v_4	🪦	k_7,v_7
					↑		

Recall that **P**$(x) = (x^2+x)/2$

Suppose we want to find the value of k_7 inside the HT and **H**$(k_7) = 5$.

H(k_7) + **P**(0) mod $N = 5$

Inserting with 🪦s

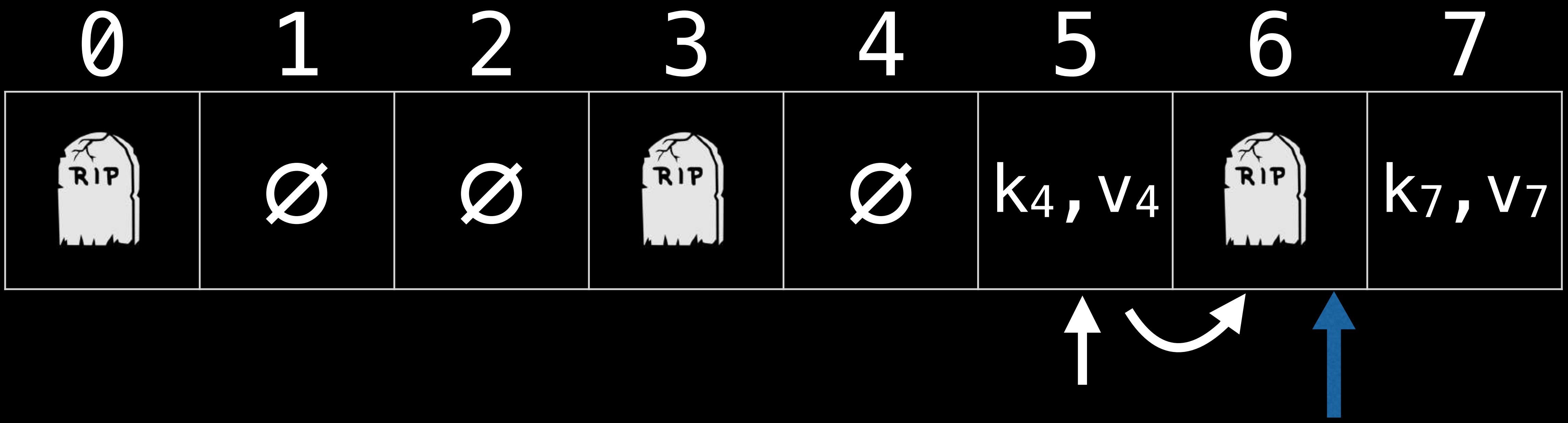
0	1	2	3	4	5	6	7
🪦	∅	∅	🪦	∅	k_4, v_4	🪦	k_7, v_7

Recall that $P(x) = (x^2+x)/2$

Suppose we want to find the value of k_7 inside the HT and $H(k_7) = 5$.

$H(k_7) + P(0) \bmod N = 5$
$H(k_7) + P(1) \bmod N = 6$

Inserting with 🪦s



0	1	2	3	4	5	6	7
🪦	∅	∅	🪦	∅	k_4, v_4	🪦	k_7, v_7

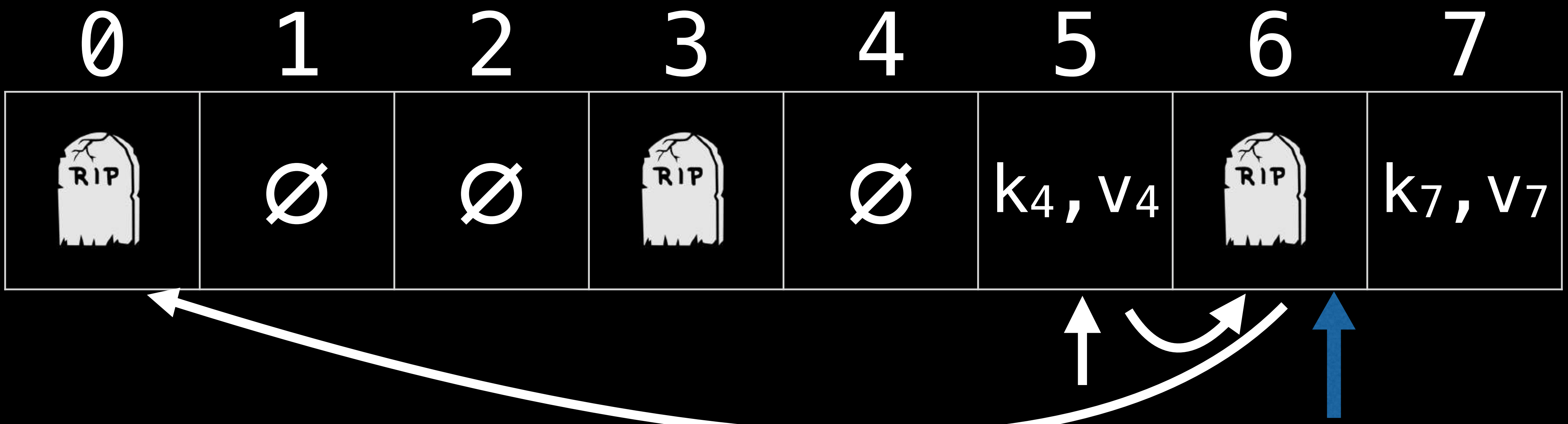
Recall that **P**(x) = (x²+x)/2

Suppose we want to find the value of k_7 inside the HT and **H**(k_7) = 5.

H(k_7) + **P**(0) mod N = 5
H(k_7) + **P**(1) mod N = 6

Position 6 is the first tombstone we encounter, so store this position for later.

Inserting with 🪦s



0	1	2	3	4	5	6	7
🪦	∅	∅	🪦	∅	k_4,v_4	🪦	k_7,v_7

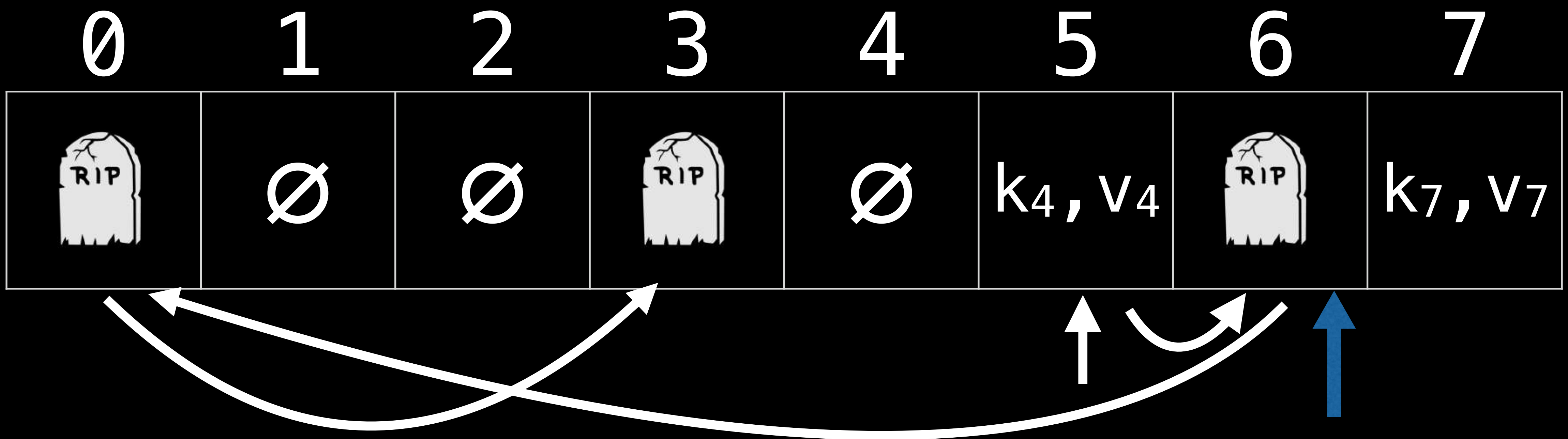
Recall that $P(x) = (x^2+x)/2$

Suppose we want to find the value of k_7 inside the HT and $H(k_7) = 5$.

$$H(k_7) + P(0) \bmod N = 5$$
$$H(k_7) + P(1) \bmod N = 6$$
$$H(k_7) + P(2) \bmod N = 0$$

Still haven't found k_7, keep probing..

Inserting with 🪦s



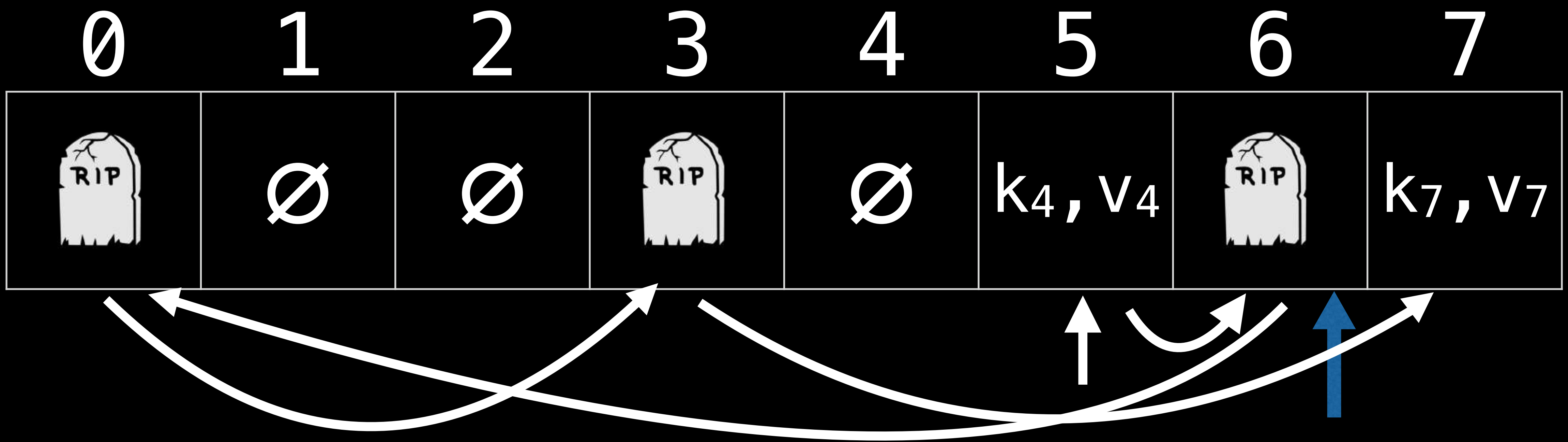
0	1	2	3	4	5	6	7
🪦	∅	∅	🪦	∅	k_4,v_4	🪦	k_7,v_7

Recall that $P(x) = (x^2+x)/2$

Suppose we want to find the value of k_7 inside the HT and $H(k_7) = 5$.

$H(k_7) + P(0) \bmod N = 5$
$H(k_7) + P(1) \bmod N = 6$
$H(k_7) + P(2) \bmod N = 0$
$H(k_7) + P(3) \bmod N = 3$

Inserting with 🪦s



0	1	2	3	4	5	6	7
🪦	∅	∅	🪦	∅	k_4,v_4	🪦	k_7,v_7

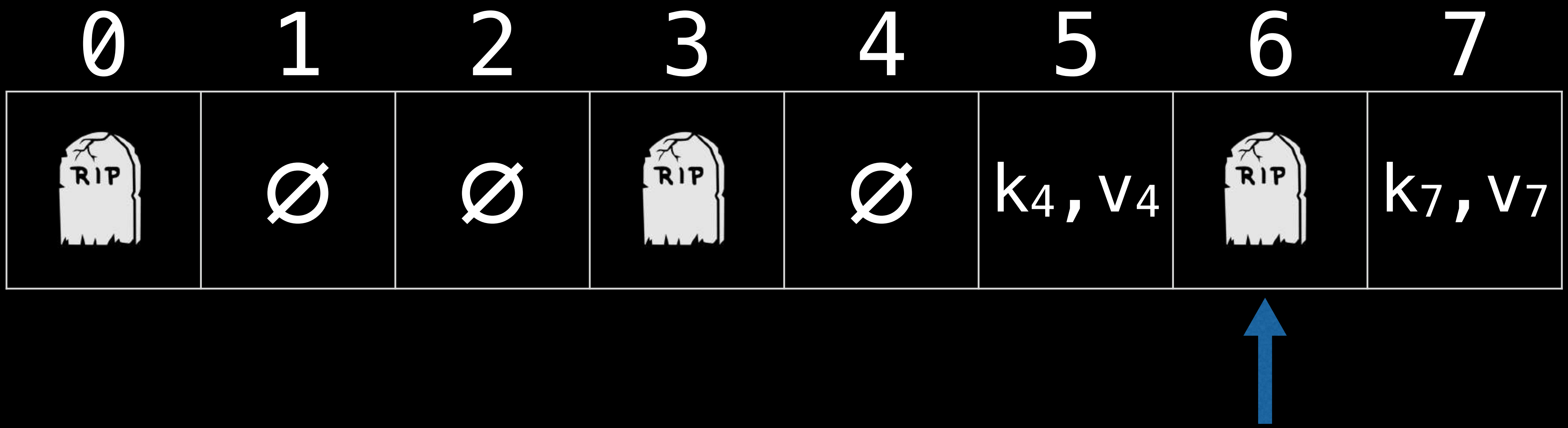
Recall that $P(x) = (x^2+x)/2$

Suppose we want to find the value of k_7 inside the HT and $H(k_7) = 5$.

$H(k_7) + P(0) \bmod N = 5$
$H(k_7) + P(1) \bmod N = 6$
$H(k_7) + P(2) \bmod N = 0$
$H(k_7) + P(3) \bmod N = 3$
$H(k_7) + P(4) \bmod N = 7$

Found it!

Inserting with 🪦s

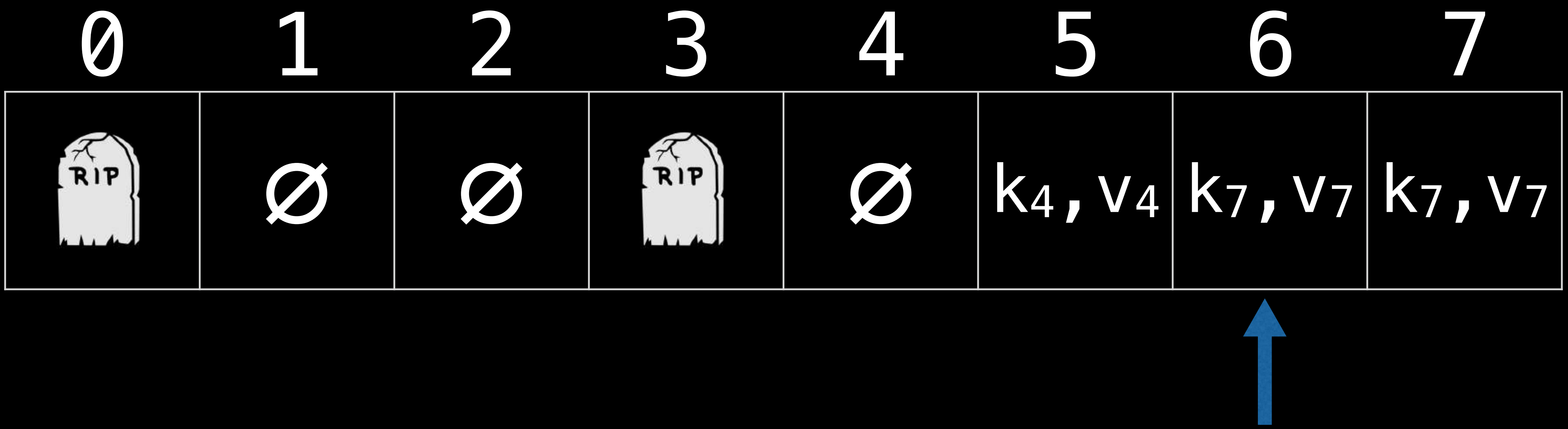


0	1	2	3	4	5	6	7
🪦	∅	∅	🪦	∅	k_4, v_4	🪦	k_7, v_7

We found the key k_7 and its associated value v_7, but we don’t want to probe an additional four times to find k_7 every time we do a lookup for its value.

An optimization we can do it replace the earliest tombstone encountered with the value we did a lookup for. The next time we lookup the key it’ll be found much faster! We call this **lazy deletion**.

Inserting with 🪦s

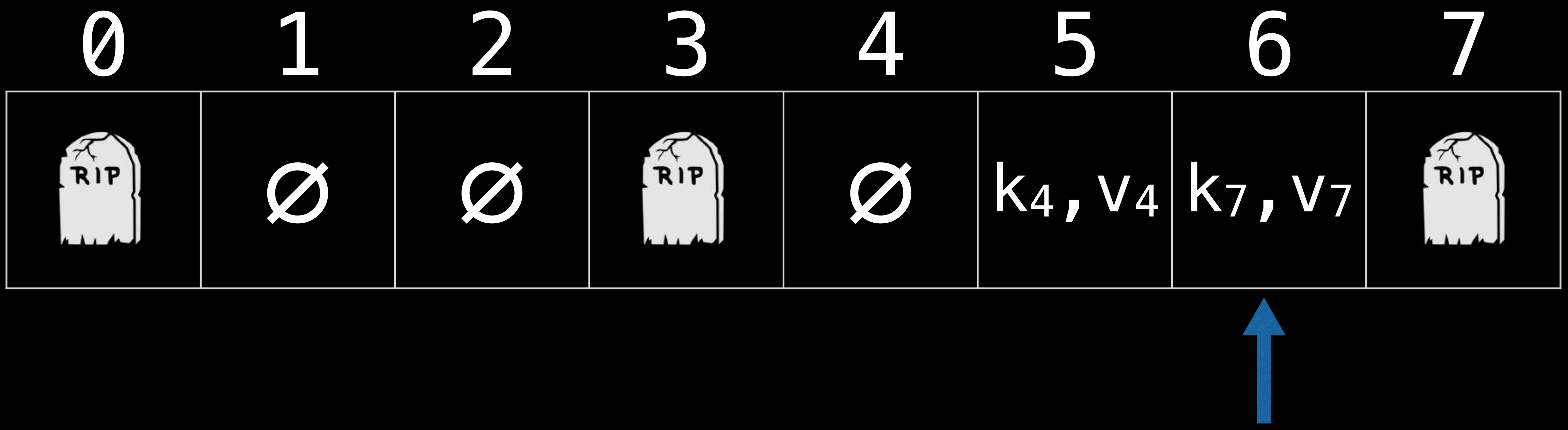


0	1	2	3	4	5	6	7
🪦	∅	∅	🪦	∅	k_4,v_4	k_7,v_7	k_7,v_7

We found the key k_7 and its associated value v_7, but we don’t want to probe an additional four times to find k_7 every time we do a lookup for its value.

An optimization we can do it replace the earliest tombstone encountered with the value we did a lookup for. The next time we lookup the key it’ll be found much faster! We call this **lazy deletion**.

Inserting with 🪦s



0	1	2	3	4	5	6	7
🪦	∅	∅	🪦	∅	k_4,v_4	k_7,v_7	🪦

We found the key k_7 and its associated value v_7, but we don’t want to probe an additional four times to find k_7 every time we do a lookup for its value.

An optimization we can do it replace the earliest tombstone encountered with the value we did a lookup for. The next time we lookup the key it’ll be found much faster! We call this **lazy deletion**.

Inserting with 🪦s

0	1	2	3	4	5	6	7
🪦	∅	∅	🪦	∅	k_4, v_4	k_7, v_7	🪦

We found the key k_7 and its associated value v_7, but we don't want to probe an additional four times to find k_7 every time we do a lookup for its value.

An optimization we can do it replace the earliest tombstone encountered with the value we did a lookup for. The next time we lookup the key it'll be found much faster! We call this **lazy deletion**.

Next Video:
hash table source code!

Multiple hash table implementations and source code and tests can all be found at:

github.com/williamfiset/data-structures

Hash table Source Code

William Fiset

Source Code Link

Implementation source code
and tests can all be found
at the following link:

github.com/williamfiset/data-structures

NOTE: Make sure you have understood the previous videos in this section explaining how a hash table works before continuing!