Data Structures and Algorithms

XMUT–COMP 103 – 2026 T1

Maps, Sorting

Felix Yan

School of Engineering and Computer Science

Victoria University of Wellington

Using Sets: Vocabulary, another requirement:

- Vocabulary:
  - Given a file of words (from a book)
  - Count the number of words and the number of distinct words.
  - Print out the vocabulary:
    - (a) all words, alphabetically
    - (b) the top 100 words (by frequency)

  - How can we sort the words?
  - How can we keep track of the frequency of words, and then sort by that?

  - We need to store each word we find PLUS how many times we have seen it.

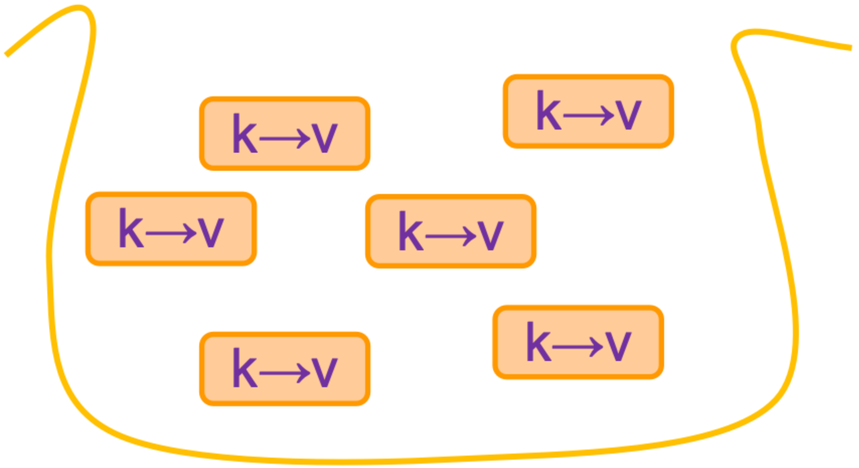
    the:50 we:8 sun:1 play:2 in:22 cat:20 hat:18

This is a "Map" — a mapping from "keys" to associated "values"
here: mapping from words to their counts

Maps

A Map is

- a collection of data with an index:
  - index –> data
- a collection of key –> value pairs
- a mapping from keys to values

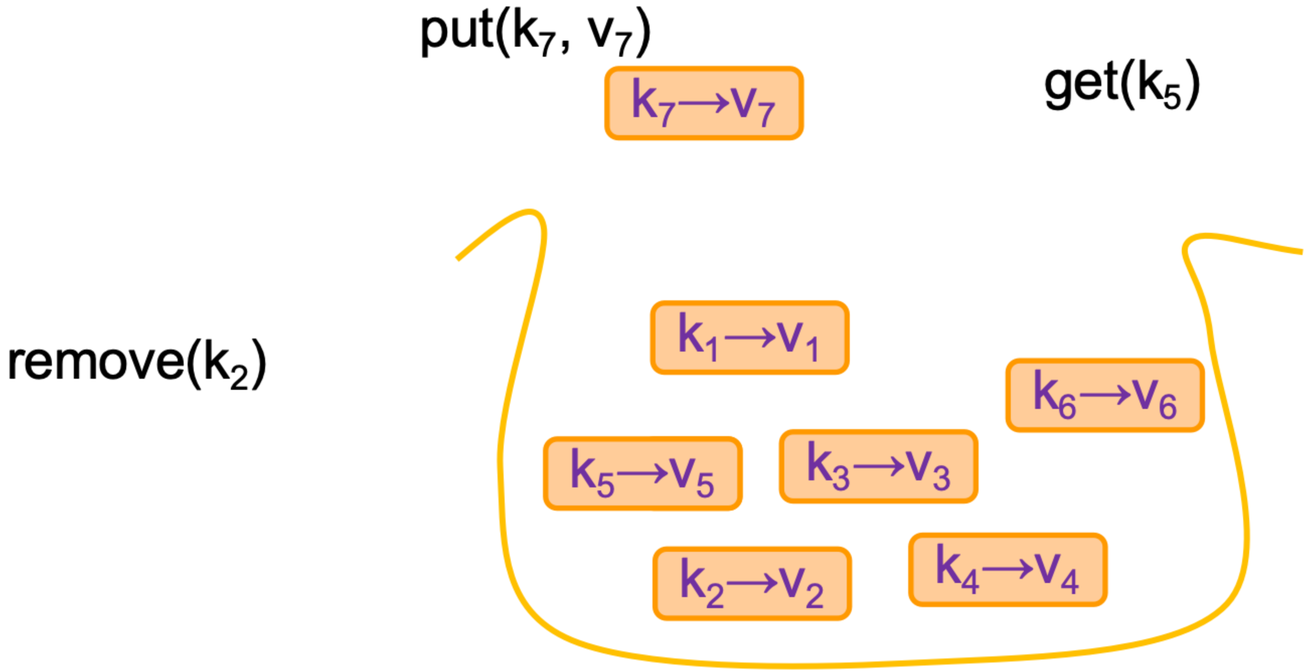


- eg: University: student ID –> student
- eg: phonebook: name –> phone number
- eg: Bank: account number –> account
- eg: Vocab: word –> count

- Note: the keys must be unique (the keys are a Set); values can be duplicated

Maps

- Fundamental operations:
  - put(key, value)
  - get(key)
  - remove(key)



Maps in Java

- Map is part of the Collections library (but a *Map* isn't a *Collection*)
- To declare a Map in Java, need two types: key and value
  - **private** Map<String, String> dictionary = **new** HashMap<String, String>();
    - *given a word, get the definition*
  - **private** Map<String, Person> members = **new** HashMap<String, Person>();
    - given a name, get the Person object for that person.
  - **private** Map<String, Integer> vocab = **new** HashMap<String, Integer>();
    - given a word, get the count of that word.
- Stores the key–value pairs in Map.Entry objects
- Implementations: HashMap, TreeMap (like HashSet and TreeSet)
  - HashMap hashes the keys to work out where to store the key–value pair
  - TreeMap uses the ordering of the keys to construct the binary search tree to store the pairs.

Maps in Java: Operations

- more operations:
  - get(key) → value (or null, if none)
  - put(key, value) → old value (or null if no old value)

  - remove(key) → value removed (or null, if none)
  - replace(key, newvalue) → old value (if no old value, doesn't put, and returns null)
  - containsKey(key),
  - containsValue(value) *(expensive! – has to search through all the pairs)*
  - clear(), isEmpty(), size()

For iterating through a Map: [can't write: **for** (?? item : myMap){....]

- keySet() → Set<keytype>
- values() → Collection<valuetype>
- entrySet() → Set<Map.Entry<keytype, valuetype>>

TreeMap – example

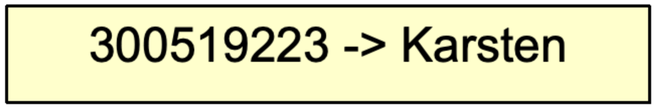
- I want to have a map that stores students preferred name using their student id as a key in a field of a class.

```
private Map<Integer, String> dictionary  = new TreeMap<Integer, String>();
```

- I start adding Karsten

```
dictionary.put(300519223, “Karsten”);
```

TreeMap – example

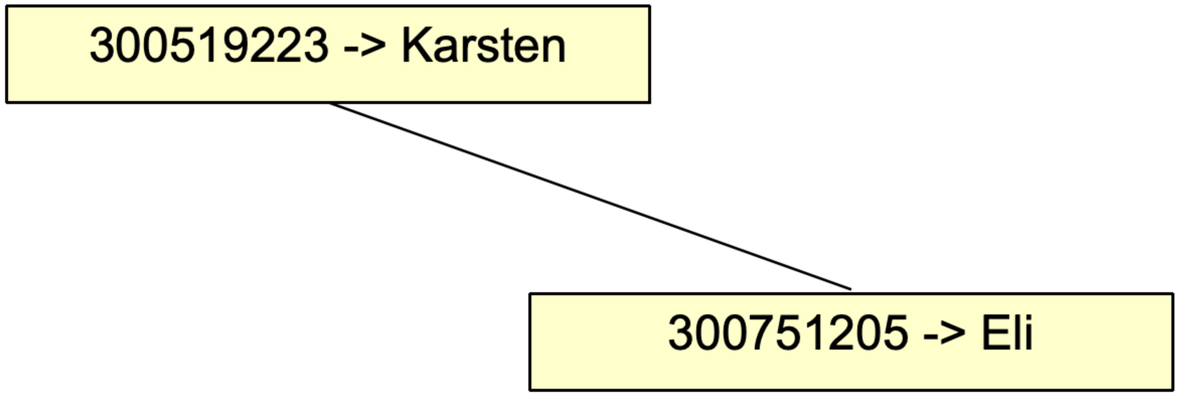


TreeMap – example

- I then add Eli

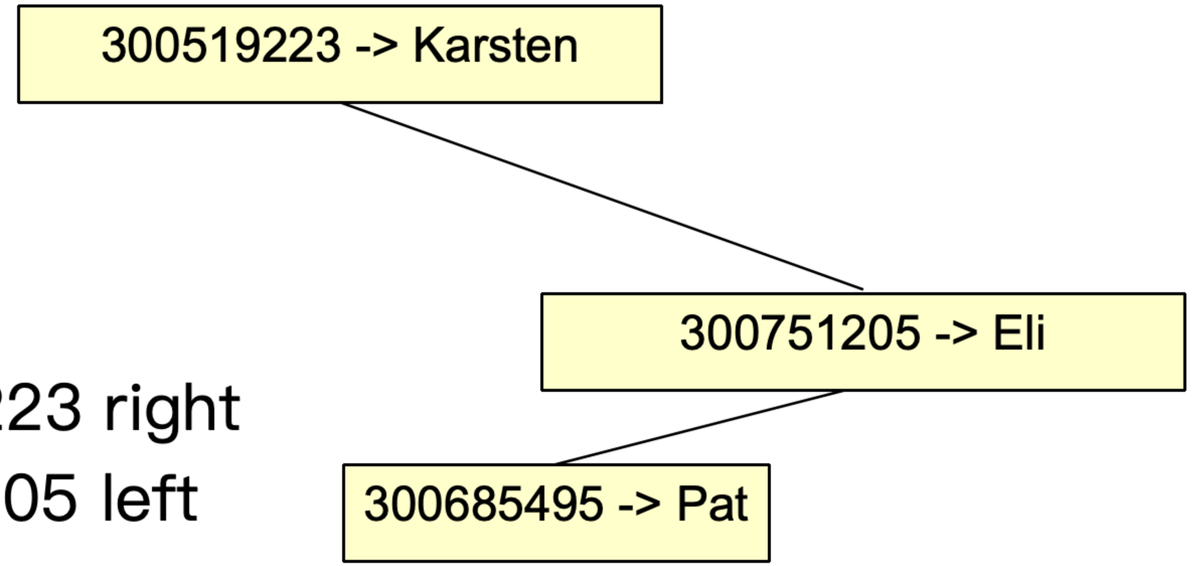
```
dictionary.put(300751205, “Eli");
```

300751205 > 300519223



300751205 > 300519223, so added on the right branch.

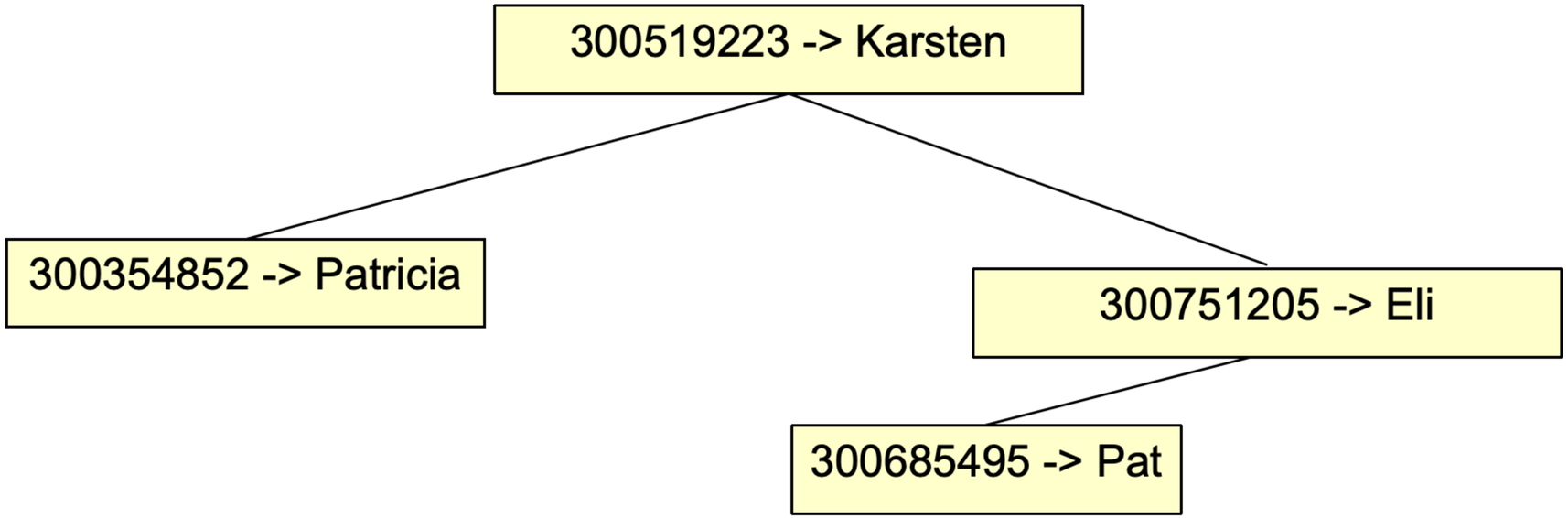
dictionary.put(300685495, “Pat");



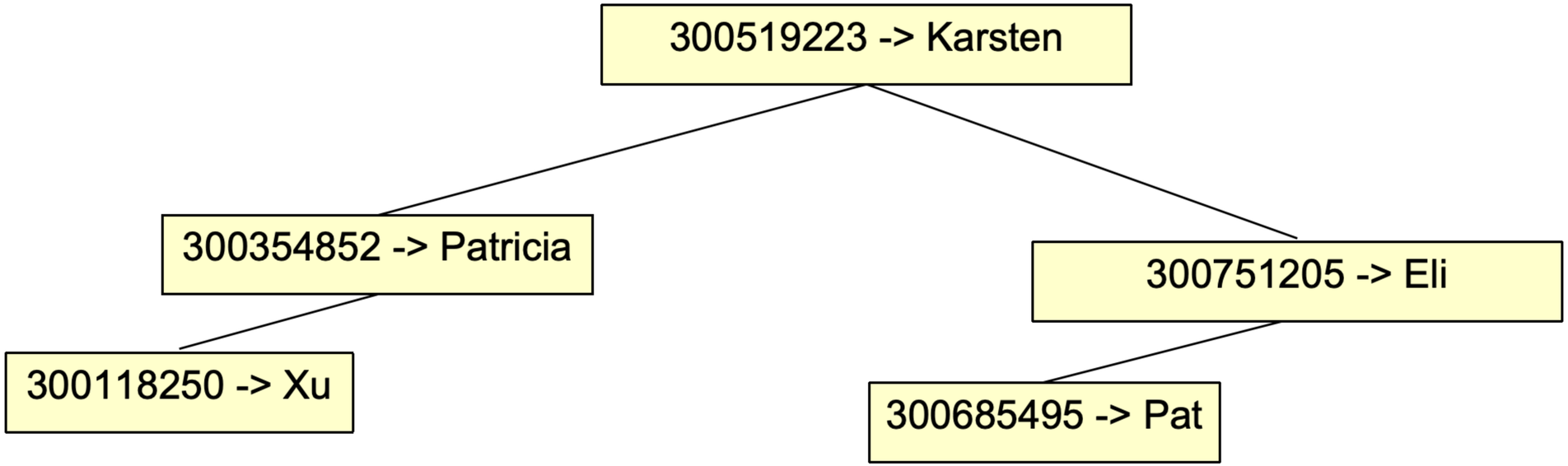
- 300685495 > 300519223 right
- 300685495 < 300751205 left

```
dictionary.put(300354852, “Patricia");

dictionary.put(300118250, “Xu");

dictionary.put(300679800, “Mark");

dictionary.put(300014172, “Joe");

dictionary.put(300424999, “Bo");

dictionary.put(300749861, “Bo"); //not the same Bo

dictionary.put(300845901, “Katherine");
```

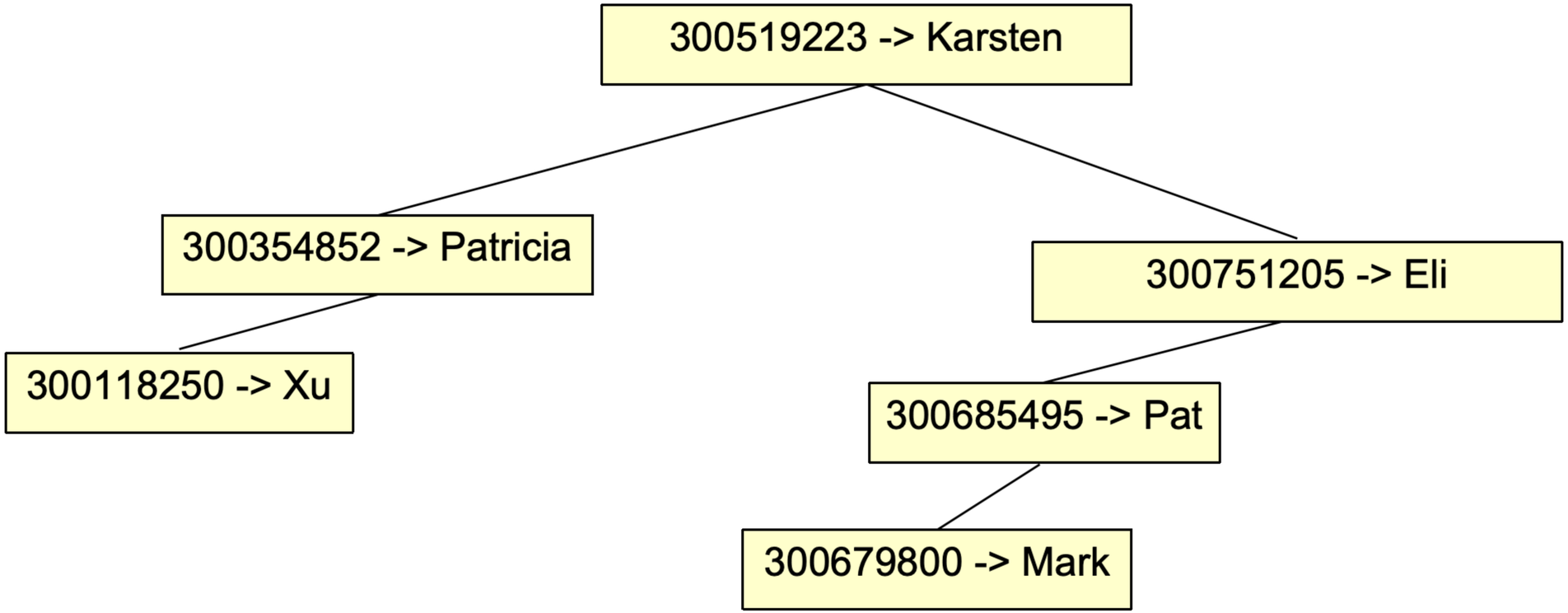
300354852 Patricia



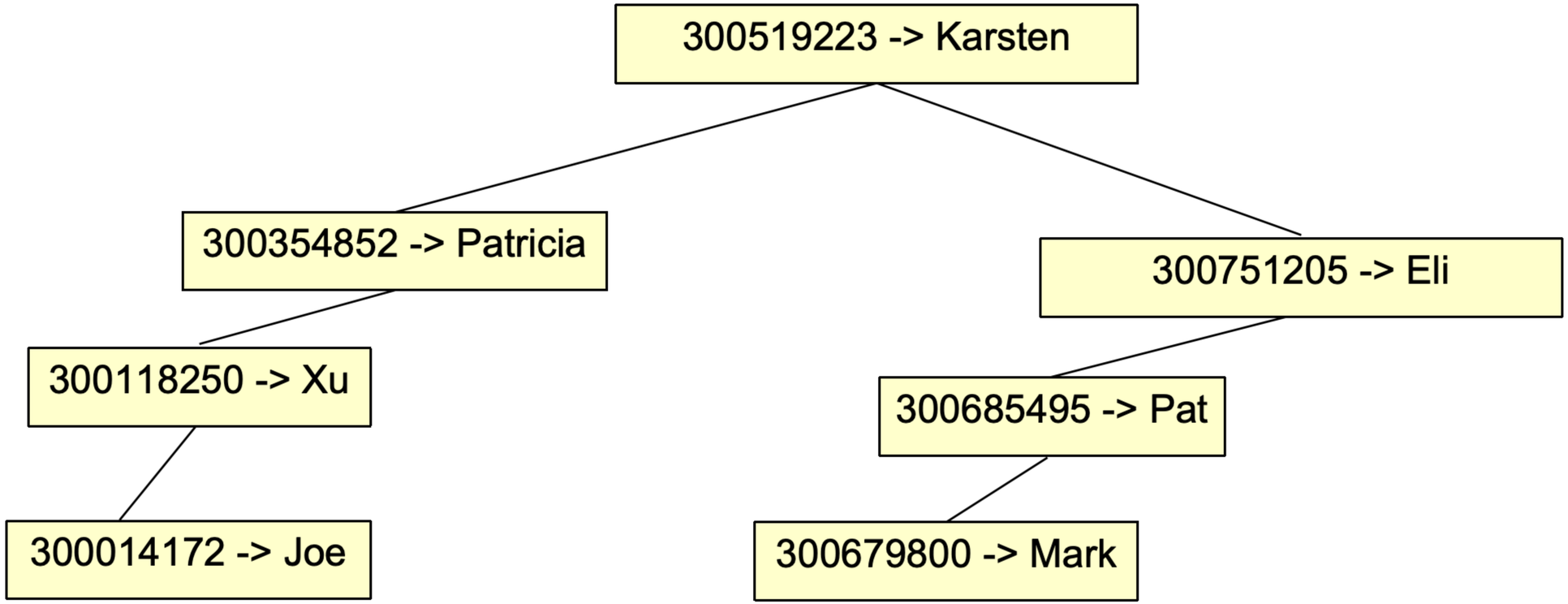
300118250 Xu



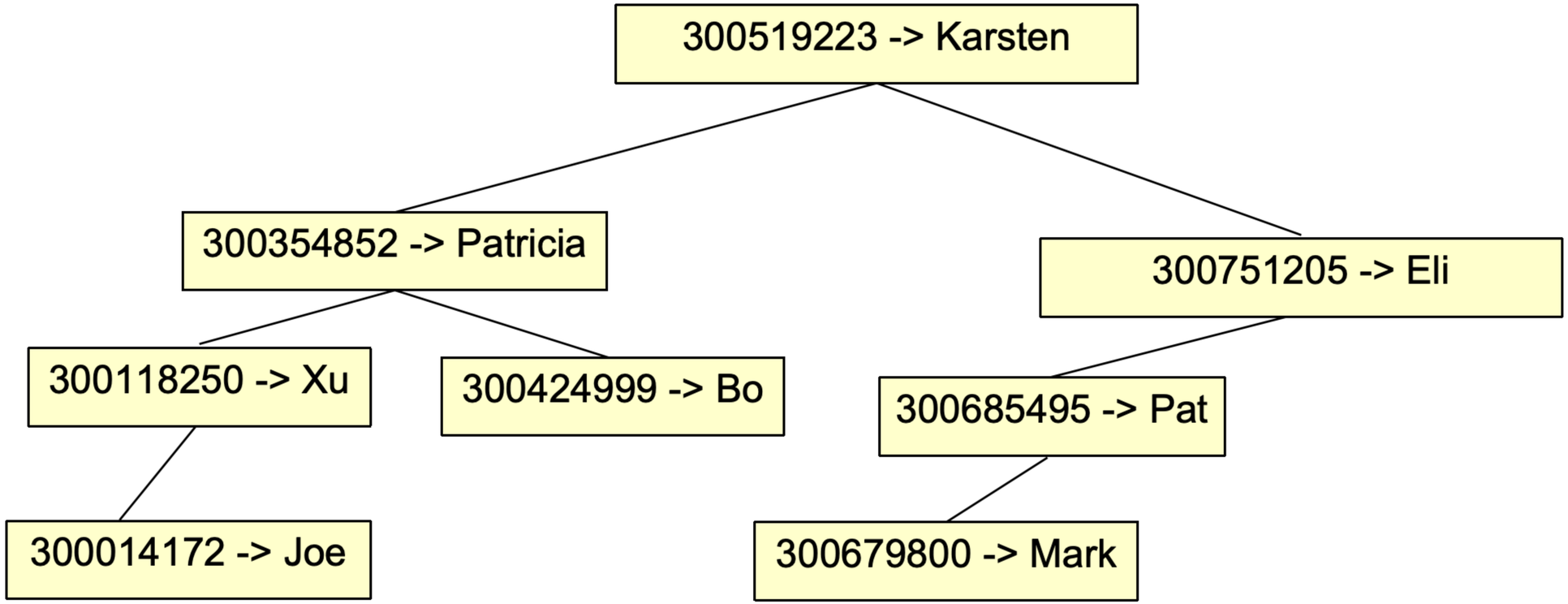
300679800 Mark



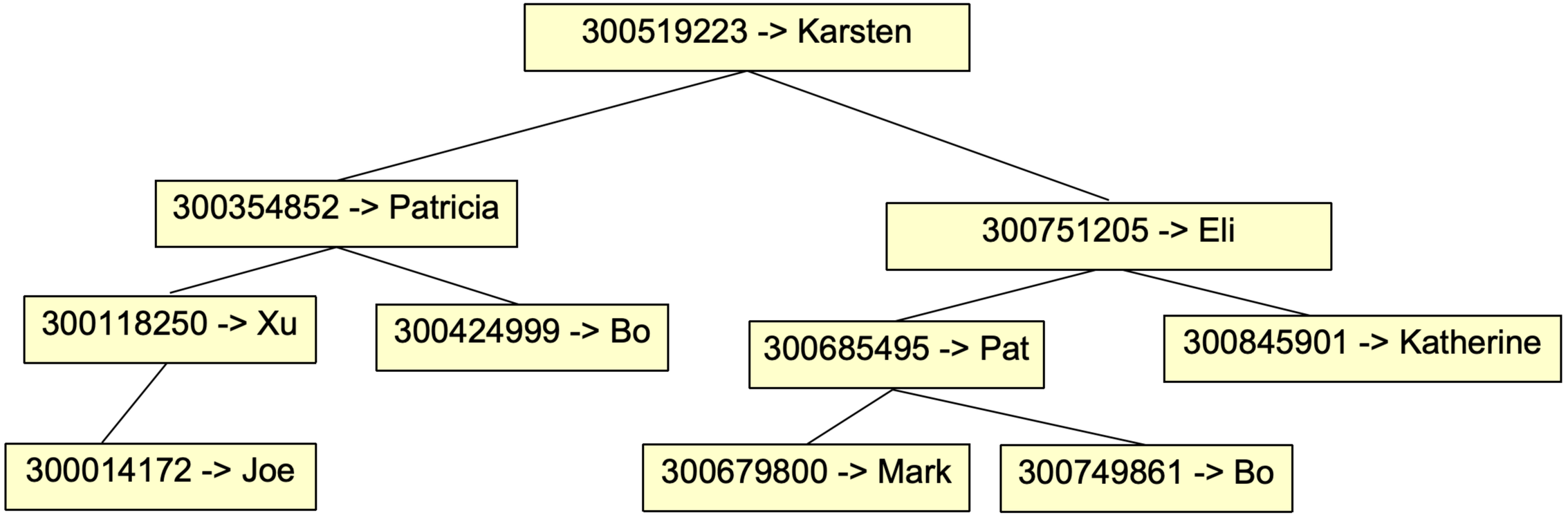
300014172 Joe



300424999 Bo

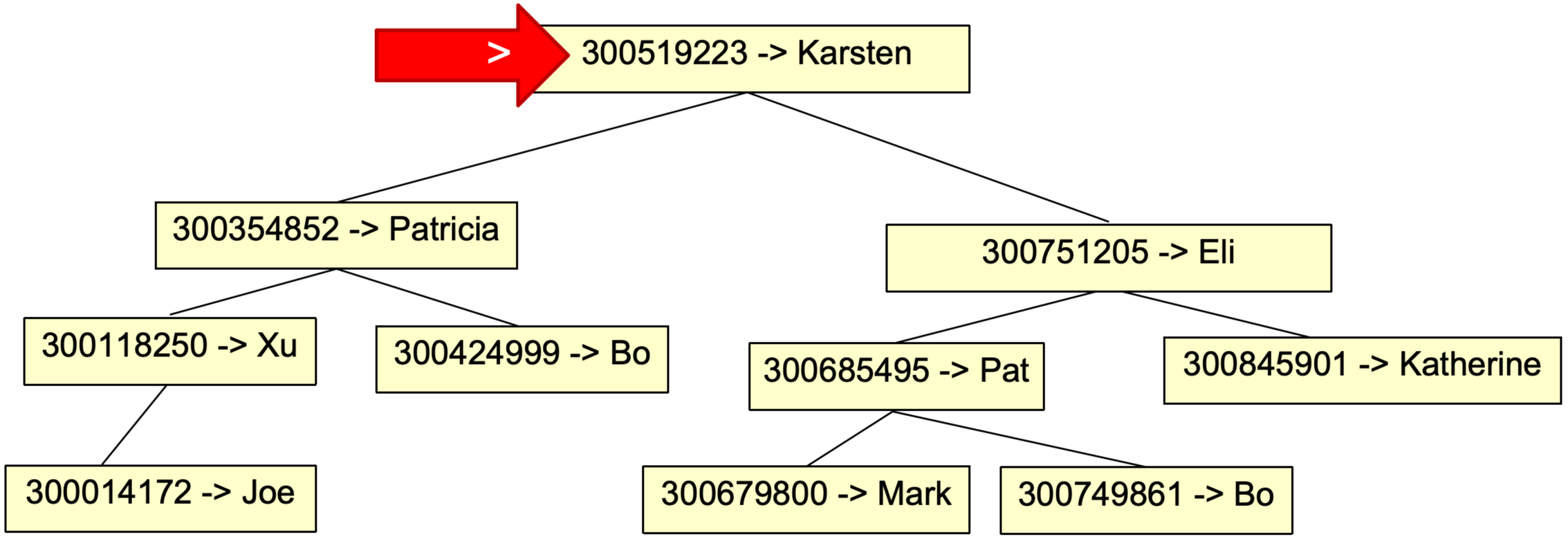


300845901 Katherine



dictionary.put(300845901, “Kat"); // Oops her preferred name is Kat

Find 300845901 and change to Kat



300519223 -> Karsten

300354852 -> Patricia

300751205 -> Eli

300118250 -> Xu

300424999 -> Bo

300685495 -> Pat

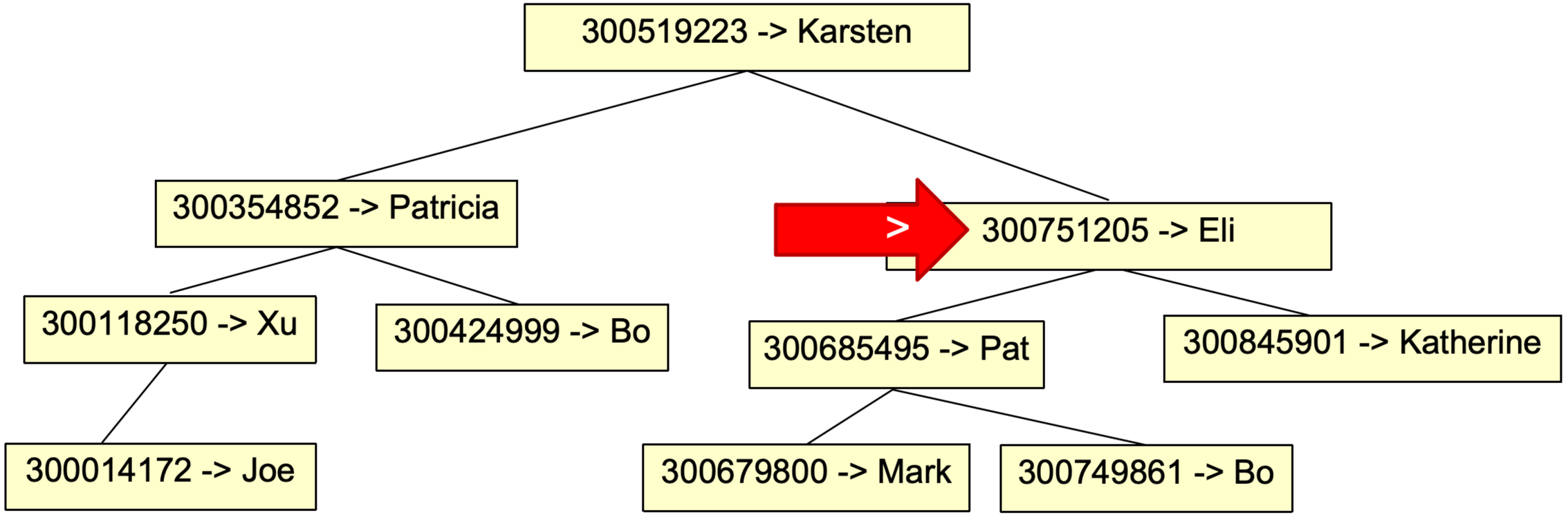
300845901 -> Katherine

300014172 -> Joe

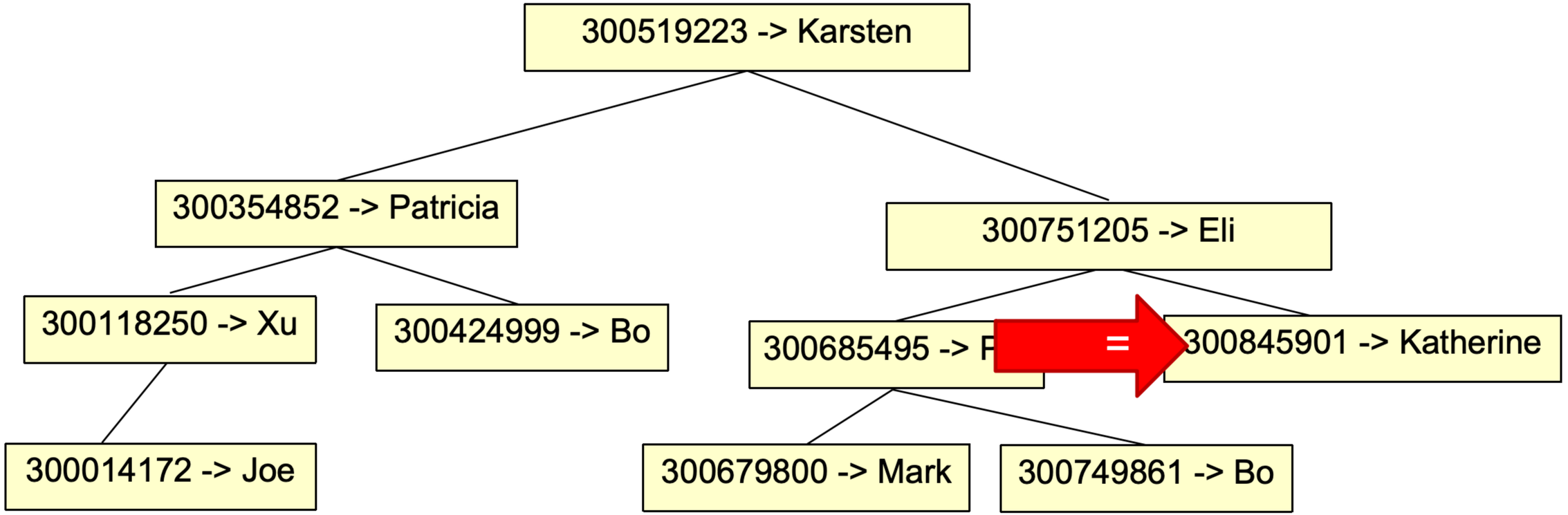
300679800 -> Mark

300749861 -> Bo

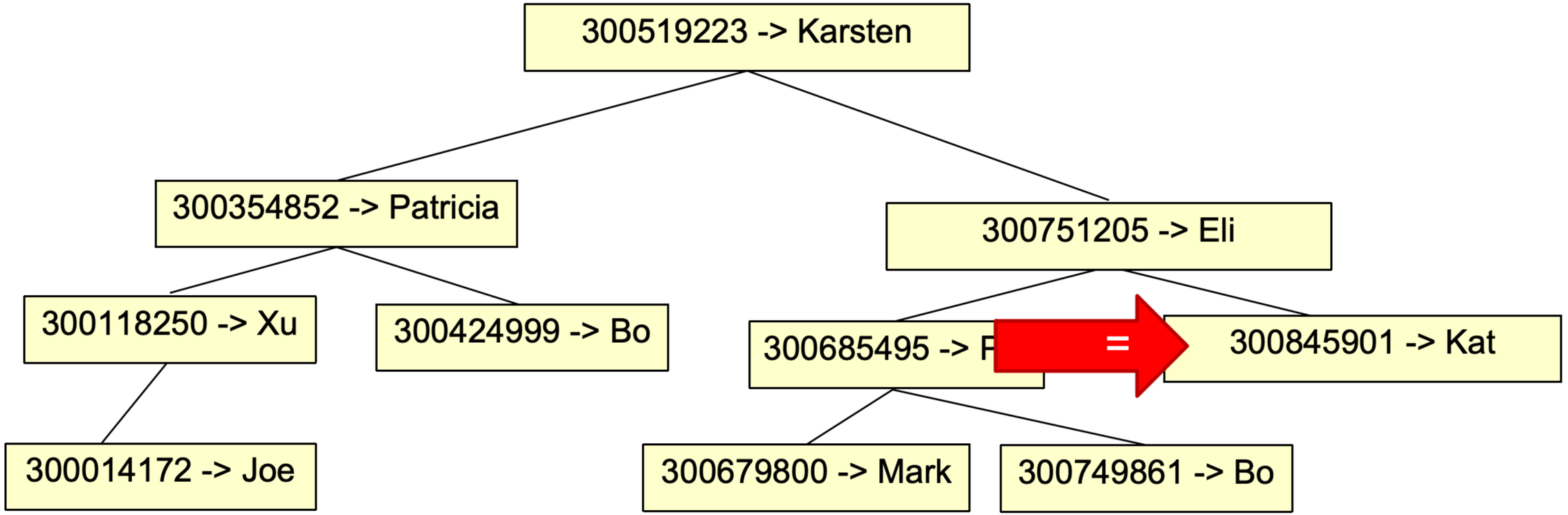
Find 300845901 and change to Kat



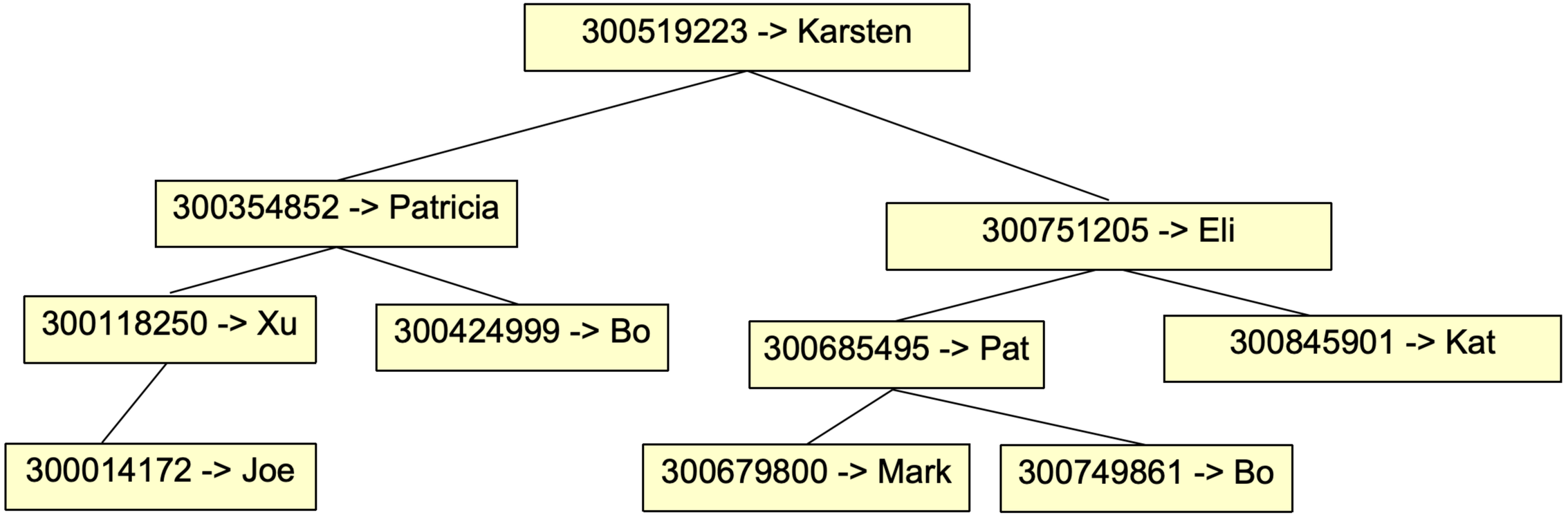
Find 300845901 and change to Kat



Find 300845901 and change to Kat

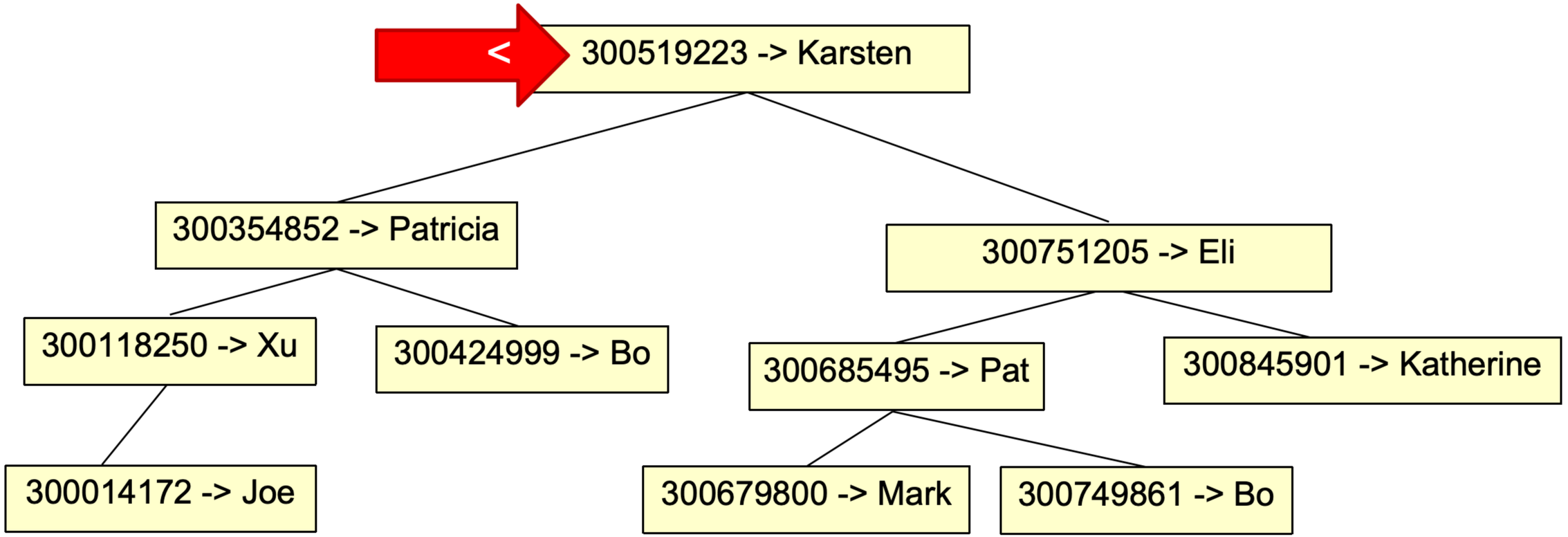


Find 300845901 and change to Kat

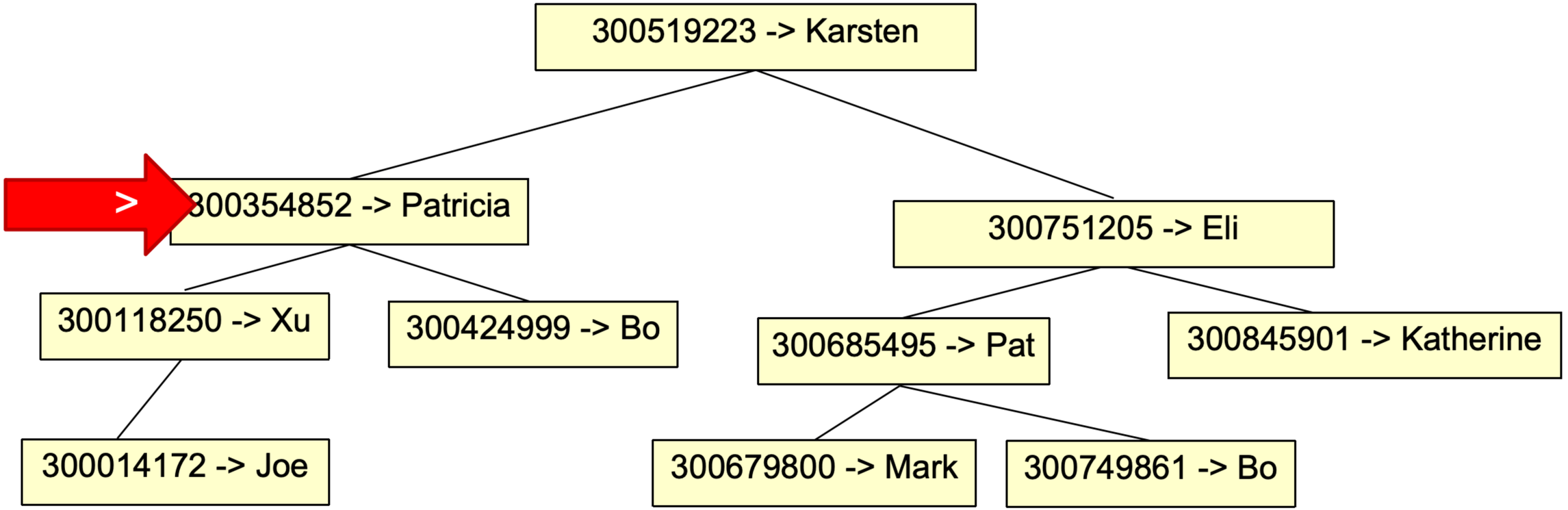


```
String  name = dictionary.get(300424999);
```

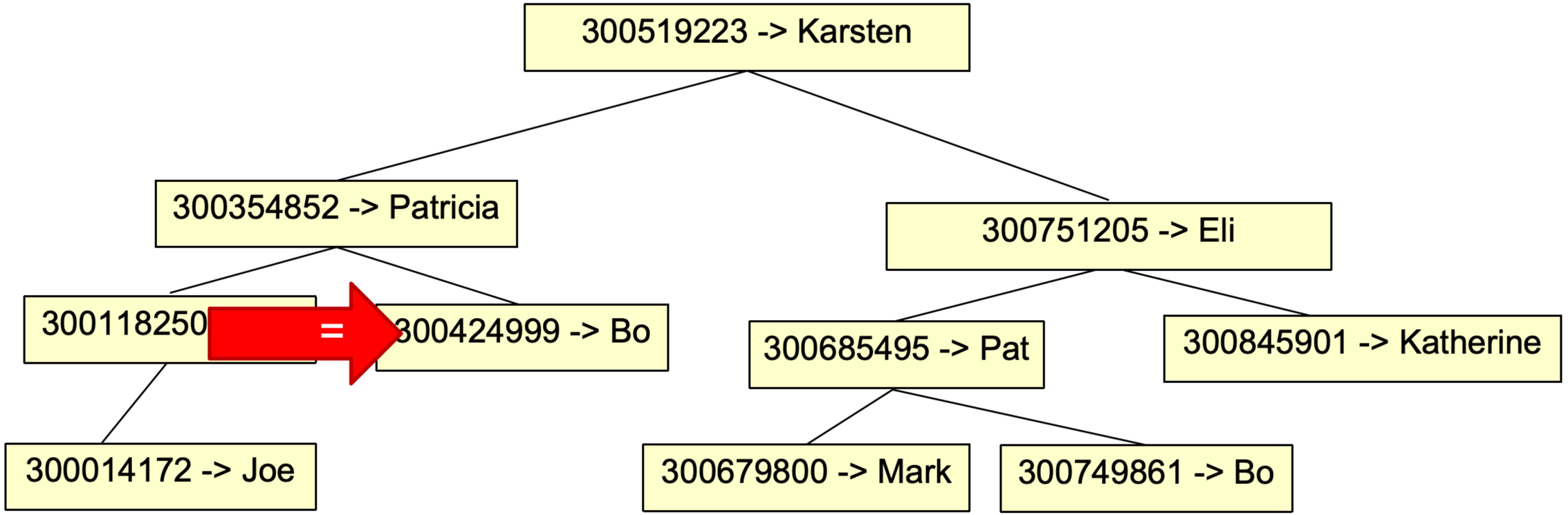
get(300424999)



get(300424999)

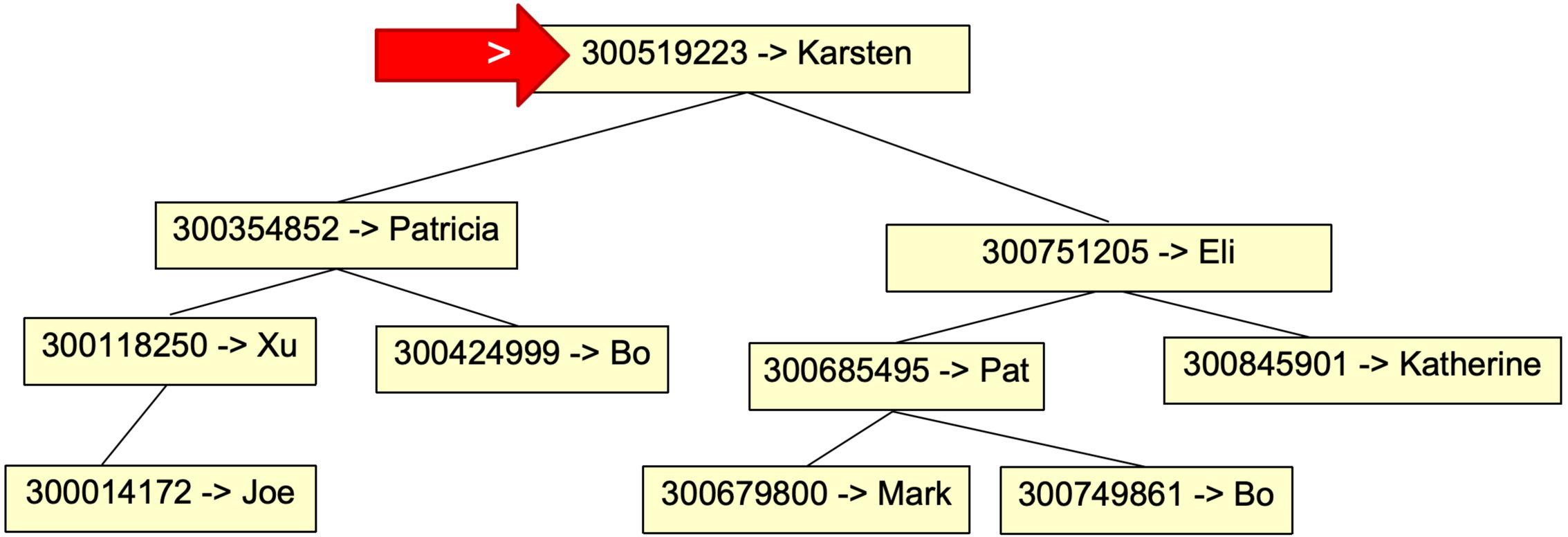


get(300424999) –> “Bo”



```
dictionary.remove(300685495);
```

remove(300685495)



300519223 -> Karsten

300354852 -> Patricia

300751205 -> Eli

300118250 -> Xu

300424999 -> Bo

300685495 -> Pat

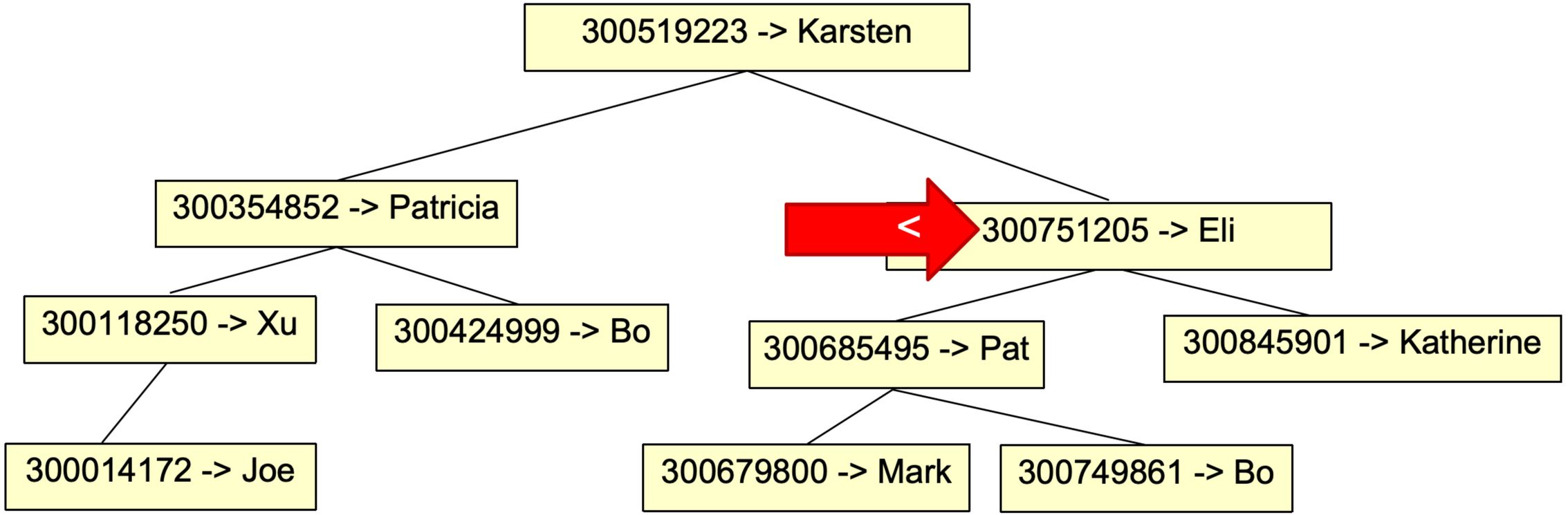
300845901 -> Katherine

300014172 -> Joe

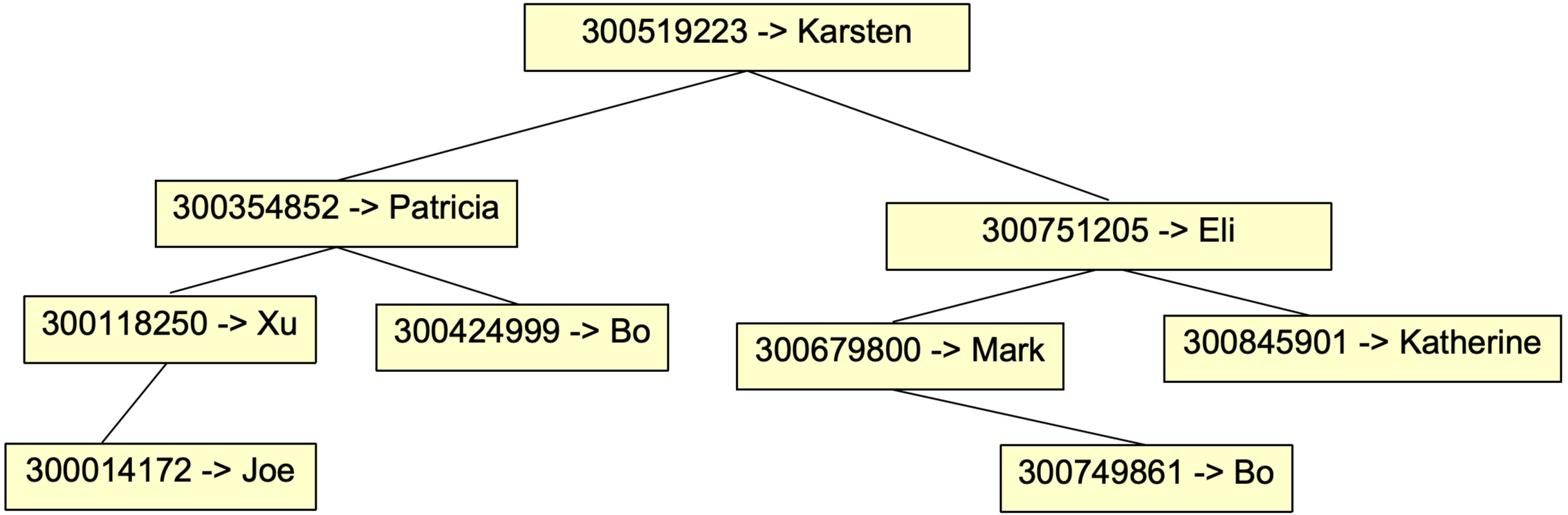
300679800 -> Mark

300749861 -> Bo

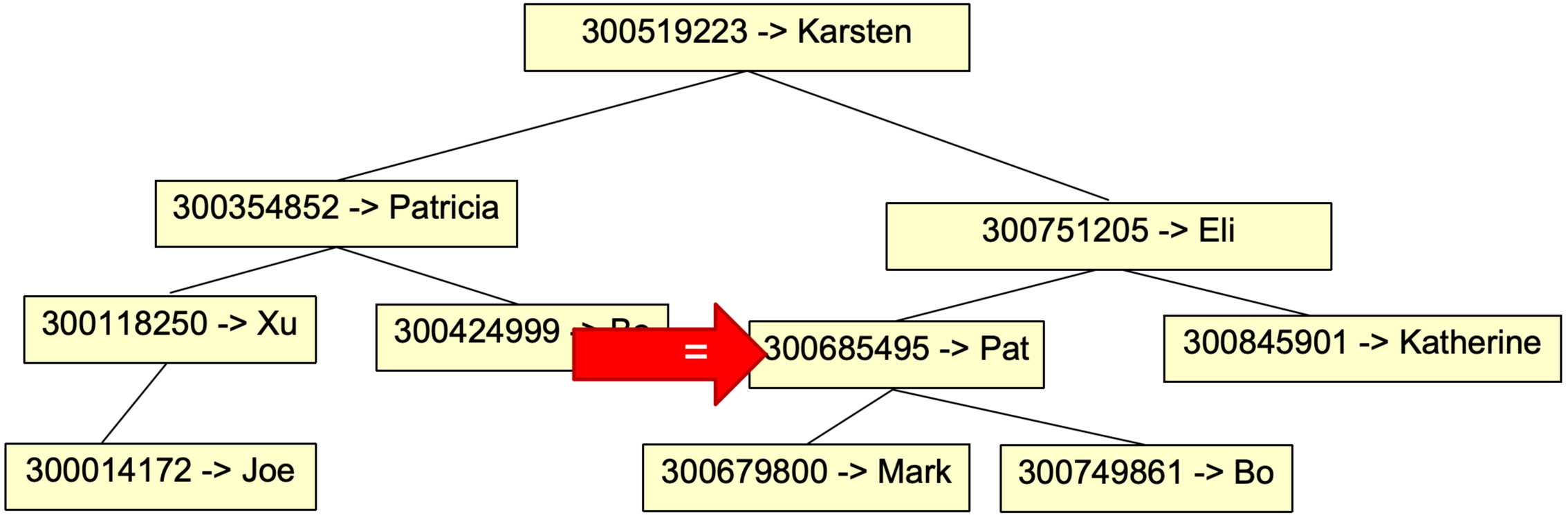
remove(300685495)



remove(300685495)



remove(300685495)



Vocabulary, yet again

- Vocabulary:
  - Given a file of words (from a book)
  - Count the number of words and the number of distinct words.
  - Print out the vocabulary:
    - (a) all words, alphabetically
    - (b) the top 100 words (by frequency)
  - How can we sort the words?
  - How can we keep track of the frequency of words, and then sort by that?

Using Sets: Vocabulary, another requirement:

- Vocabulary:
  - Given a file of words (from a book)
  - Count the number of words and the number of distinct words.
  - Print out the vocabulary:
    - (a) all words, alphabetically
    - (b) the top 100 words (by frequency)

  - How can we sort the words?
  - How can we keep track of the frequency of words, and then sort by that?

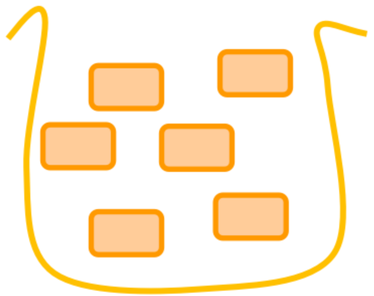
  - We need to store each word we find PLUS how many times we have seen it.

    the:50 we:8 sun:1 play:2 in:22 cat:20 hat:18

This is a "Map" — a mapping from "keys" to associated "values"
here: mapping from words to their counts

Counting Occurrences

- Vocabulary count:
  - Given a file of words (from a book)
  - Count the number of occurrences of each distinct word.



    - open the file
    - initialise Vocab = new Map <String, Integer>()
    - for each word in the file
      - if the word is not in the vocab, then
        - put(word, 1)
      - else
        - put(word, get(word)+1)
    - close the file

    - sort vocabulary by count and print out first 100 words ??????

Building the vocab map.

```
Map<String, Integer> vocab = new HashMap<String, Integer>();
Scanner sc = new Scanner(Path.of(filename));
while (sc.hasNext()){
    String word = sc.next();
    if (! vocab.containsKey(word)){
        vocab.put(word, 1);
    }
    else {
        vocab.put(word, (vocab.get(word)+1) );
    }
}

//  sort vocabulary by count and print out first 100 words     ??????
```