The ADT Map

Pearson

The ADT Map

- **Map entries contain two parts**
  - ***Keyword***
    - a search key
  - ***Value***
    - associated with key
    - desired value (or payload)

Pearson

The ADT Map

Search Keys

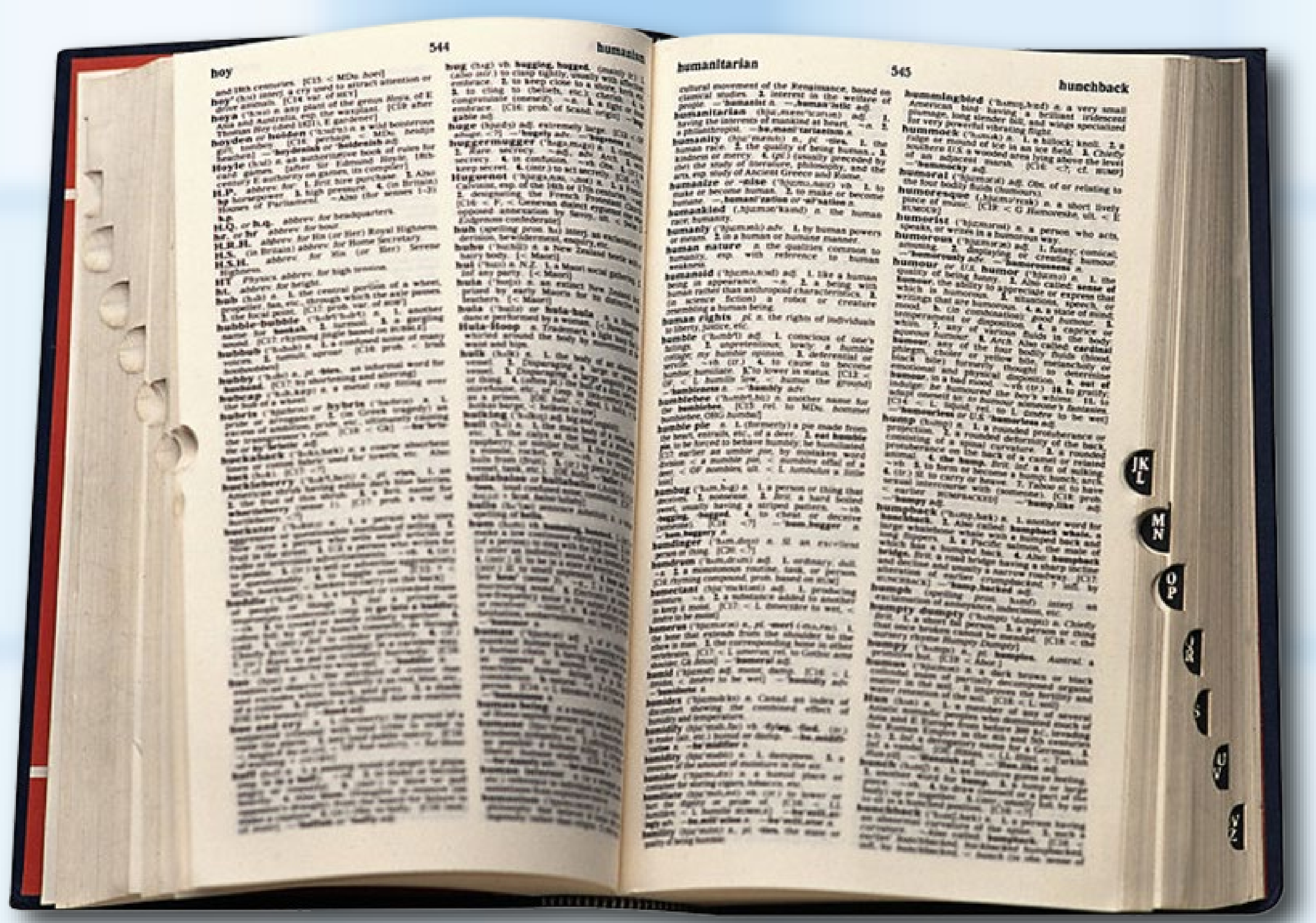
backtracking	A problem-solving strategy that, when it reaches an impasse, retraces its steps in reverse order before trying a new sequence of steps.
base case	The known case in either a recursive definition or an inductive proof. Also called the basis or degenerate case.
base class	A class from which another class is derived. The derive base class's members. Also called the ancestor class or
behavior	An action that an object performs.
binary file	A file whose elements are in the computer's internal rep binary file is not organized into lines.
box trace	A systematic way to trace the actions of a recursive function.
circuit	A special cycle that passes through every vertex (or edge) in a graph exactly once.
client	The program, module, or ADT that uses a class.
compiler	A program that translates a program written in a high-level language, such as C++, into machine language.

Values

Maps can sort entries based on the search keys, though it is not required.

The ADT Map

- **Map operations are same as other ADTs and databases**
  - add, remove, retrieve, search and traverse
  - but specified differently -
    - items are identified by search keys, not positions



- **Design Decision**
  - Distinct Search Keys
    - **`add`** could fail with duplicate key or replace current value with new value at the key
    - other methods are simpler
  - Duplicate Search Keys
    - **`add`** always adds the key, value pair
    - **`remove`** and **`getValue`** must decide which of the duplicates to remove or retrieve

Pearson

The ADT Map

- **Abstract Base Class `MapInterface`**
  - Assume that all items in the Map have distinct search keys.
  - Options for Add Functionality:
    - **Add operation can deny an attempt to insert a new entry whose search key already exists in the Map**
    - Add can replace an existing entry whose search key matches the search key of a new entry with the new entry

```
template<class ItemType, class KeyType>
class MapInterface
{
public:
   virtual bool add(const KeyType& searchKey,
                                          const ItemType& newItem) = 0;

   virtual bool remove(const KeyType& searchKey) = 0;

   virtual bool isEmpty() const = 0;

   virtual int getNumberOfItems() const = 0;

   virtual void clear() = 0;

   virtual ItemType getItem(const KeyType& searchKey) const = 0;

   virtual bool contains(const KeyType& searchKey) const = 0;

   virtual void traverse(void visit(ItemType&)) const = 0;

   virtual ~MapInterface() {  }

}; // end MapInterface
```

Map Implementations

AN ARRAY-BASED MAP

directory
José Gomez 555-1234
John Jones 555-1111
Jill Smith 555-9999
Must Synchronize Array Indices
Double the Unused/Wasted Space! (Double the entries to copy when expanding!)
keys
0 1 2 3 4 5 6 7
Array Index
Map Keys
José Gomez
John Jones
Jill Smith
555-1234
555-1111
555-9999
Parallel Arrays
numberOfEntries
3
values
Map Values
Array Index

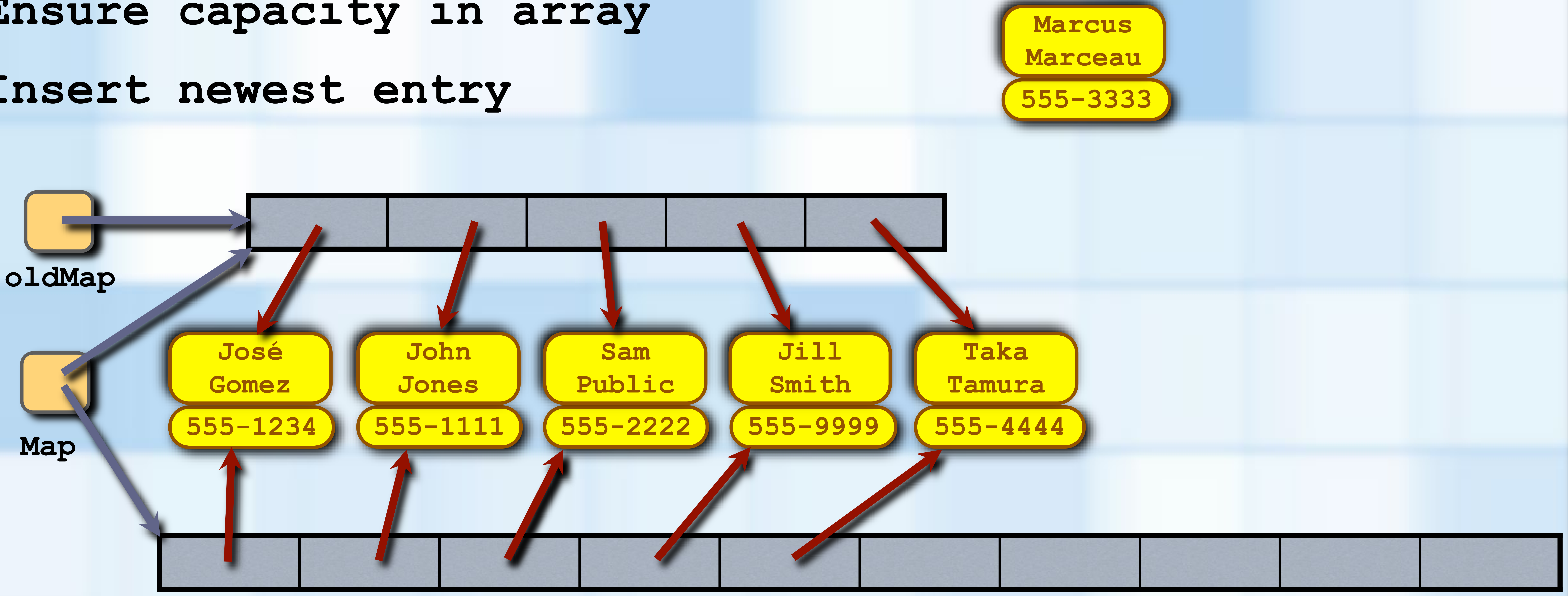
An Array-Based Map

directory
José Gomez
555-1234
John Jones
555-1111
Jill Smith
555-9999
Jill Smith
555-9999
key
value
John Jones
555-1111
key
value
José Gomez
555-1234
key
value
Entry<K,V>
numberOfEntries
3
Map
Map Entries
0
1
2
3
4
5
6
7
Array Index

Pearson

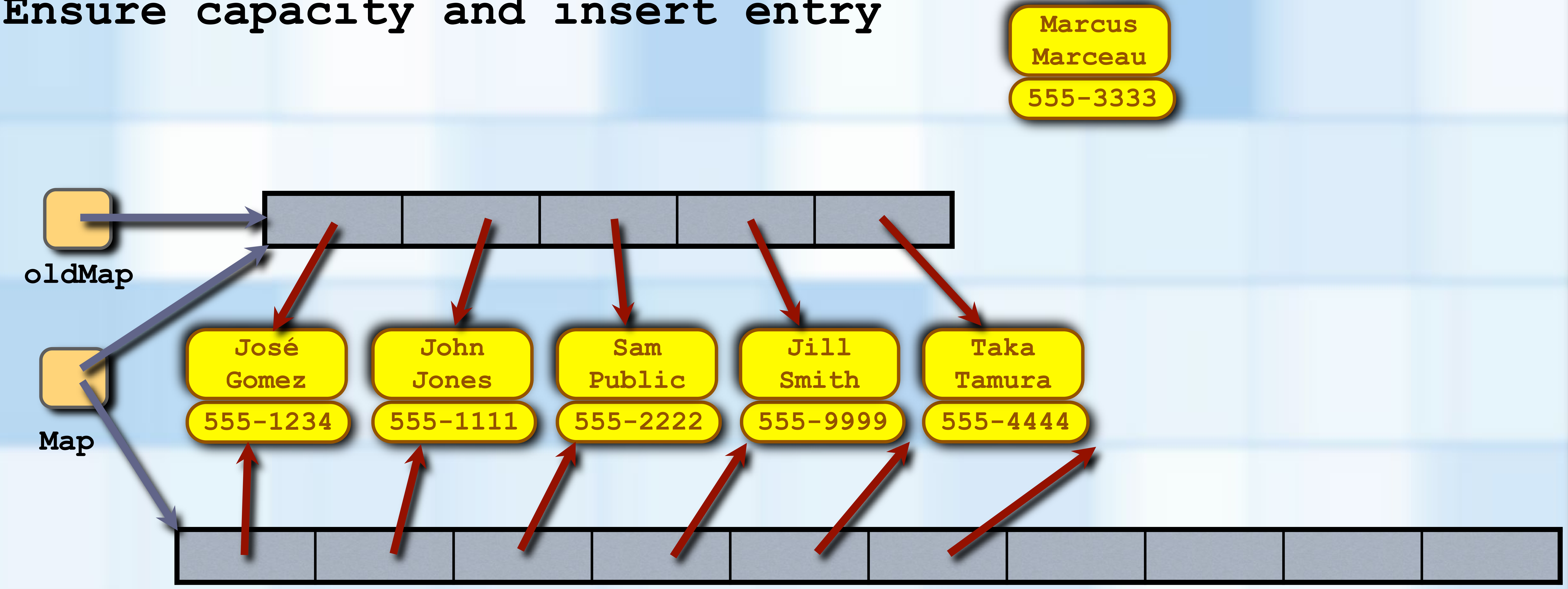
An Array-Based Map

- Ensure capacity in array
- Insert newest entry



An Array-Based Map

- Ensure capacity and insert entry



A Sorted Linked Map

directory
José Gomez
555-1234
John Jones
555-1111
Jill Smith
555-9999
head
numberOfEntries
3
José Gomez
555-1234
key
value
next
John Jones
555-1111
key
value
next
Jill Smith
555-9999
/
key
value
next
MapNode<K,V>