CSC 1052 – Algorithms & Data Structures II:

Comparable Objects

Professor Henry Carter
Spring 2017

Recap

- Collections store and access data items by their content
  - No indexing, no ordering
- Array-based implementations are simple but inefficient
  - The find() helper makes implementing the rest of the methods easy
- Vocabulary density measures the writer's use of varying words
- Vocabulary density can be measured with the help of a collection
  - Although not required, data elements can be limited to unique items if the application calls for it

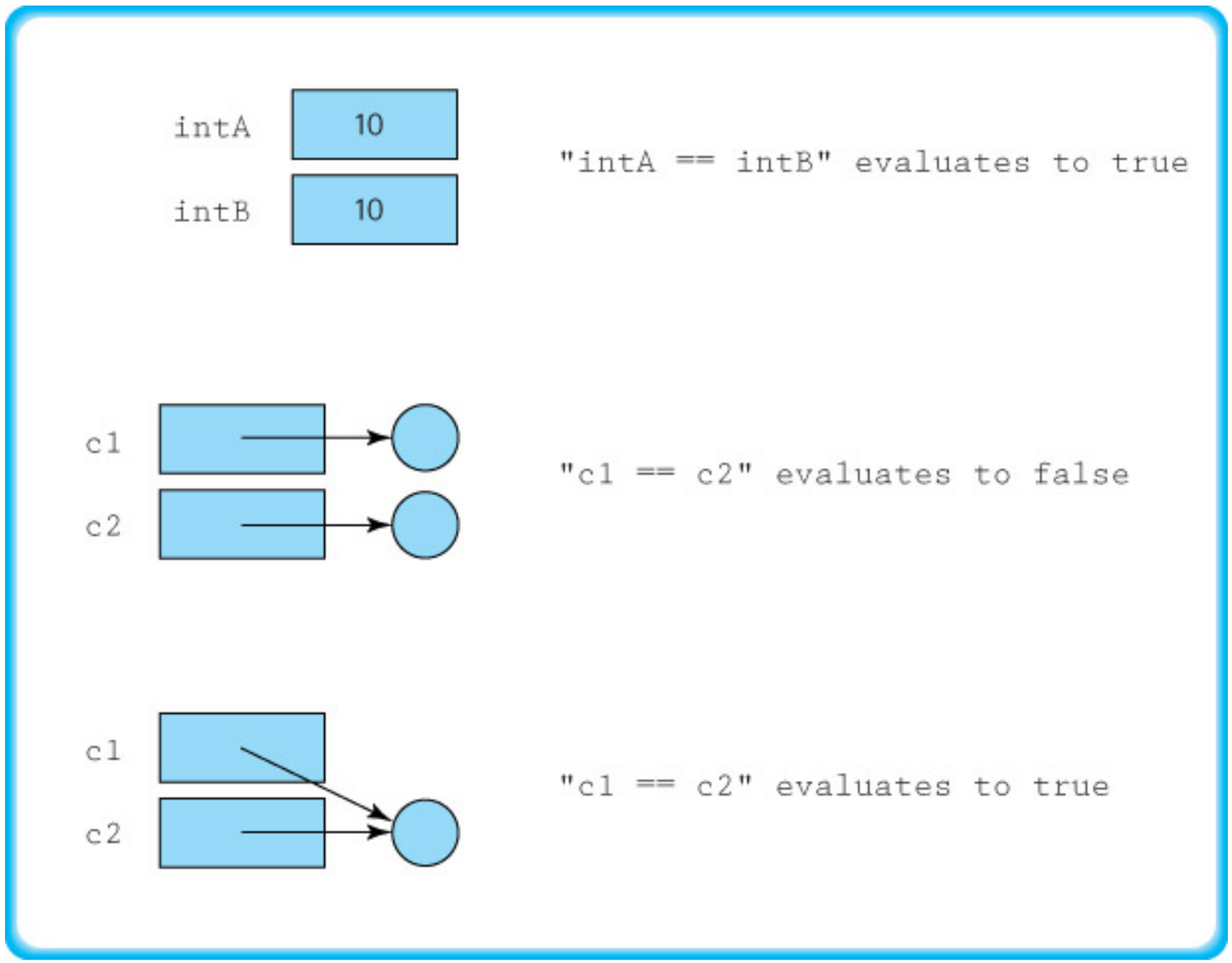
Comparison

- The Collection ADT requires elements to be equal or not
- For primitive types (and some simple objects) this concept is simple
- How do we define equality for more complex objects?

equals()

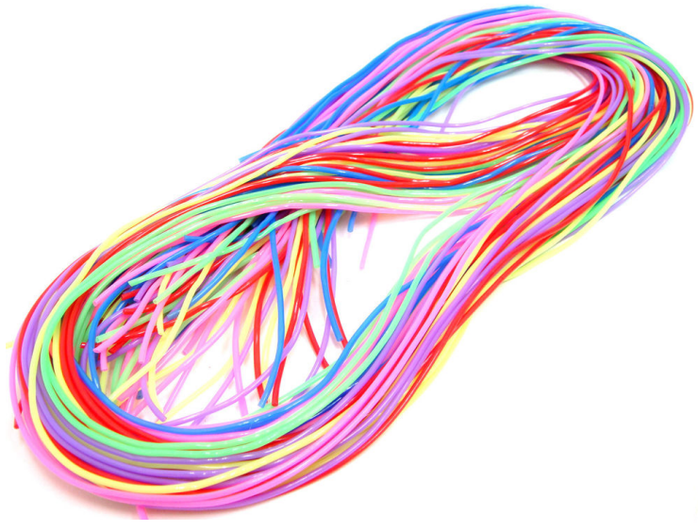
- Recall: '==' compares the values stored at the variable on either side
  - Compares value for primitive types
  - Compares pointers for objects
- The equals() method is defined for all objects
  - By default, it works like '=='
- Overriding this method allows us to define equality for any object

Equals versions



How do we define equals()?

- Strings?
- Circles?
- Bank accounts?
- Famous people?



Keys

- For any object needing comparison, we must define "key" information
- The key is the unique identifier of the object
- May be part or all of the data stored in the object

- Examples:
  - String?
  - Famous Person?

Example: Famous Computer Scientists

- Initializes a collection of "famous person" objects
- Polls the user for names
- Reports information about the queried name

Setup and run

- Download and set up the program
- Run for the following queries
  - Ada Lovelace
  - Alan Turing
  - Grace Hopper
- Open the code

Highlights

- Equals() only compares names
- This allows us to search for names without having an object that matches in *every* class variable
- When we call person = people.get(person)
  - We replace the old version: "Ada", "Lovelace", 0, ""
  - With the new: "Ada", "Lovelace", 1815, "Considered by many to be the first programmer"

The Comparable Interface

- Given that we can define equality for objects, we can also define inequality
- The Comparable interface requires one method to be implemented: compareTo()
- Implementing this interface lets users know that this class has a defined way to rank high-low

compareTo()

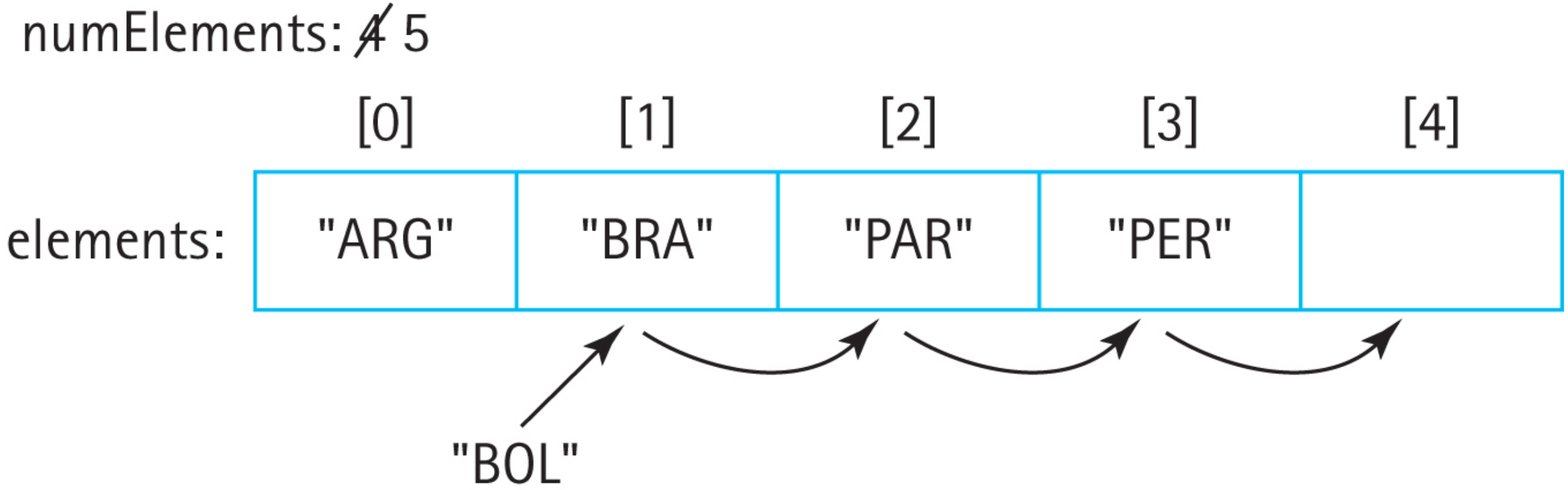
- When called on an object:
  - obj.compareTo(compareObj)
  - Returns
    - 0 if they are equal
    - Negative if obj < compareObj
    - Positive if obj > compareObj

How does this help?

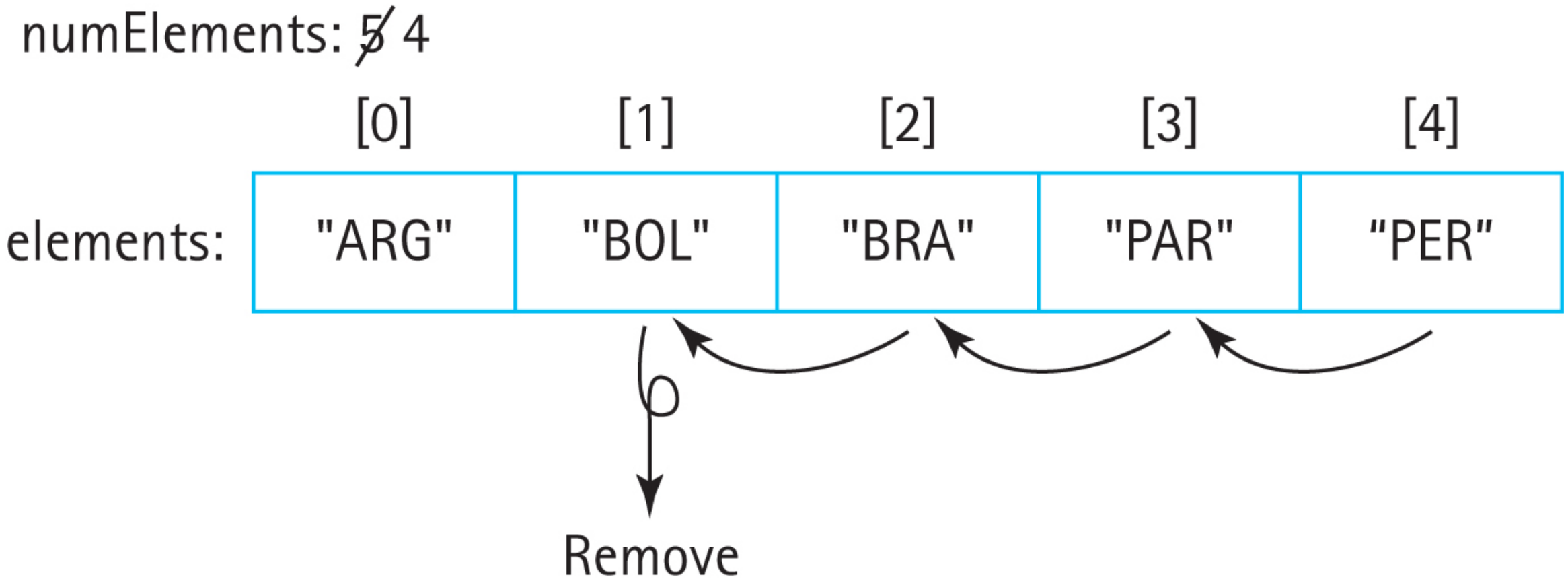
- Sorting!
- Storing collection elements in a sorted array will:
  - Increase the cost of insertion and deletion
  - Decrease the cost of searching

- Which is more important to your application?

Add

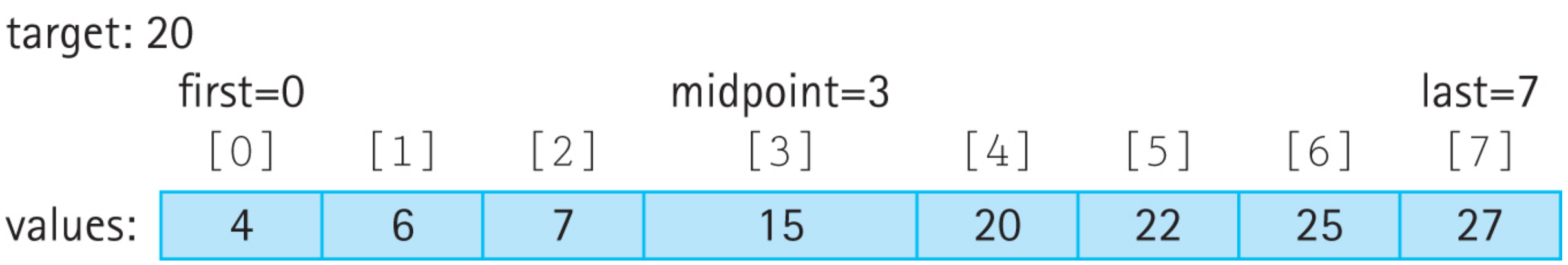


Remove

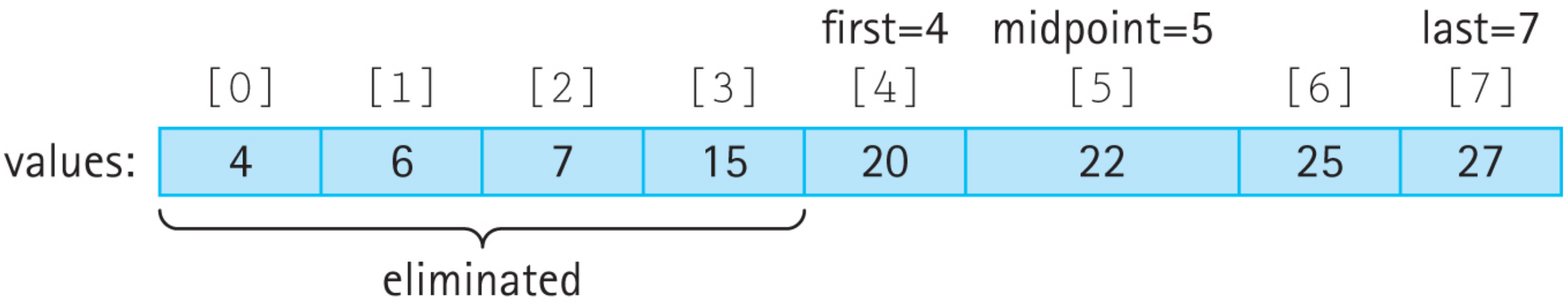


Find

target: 20



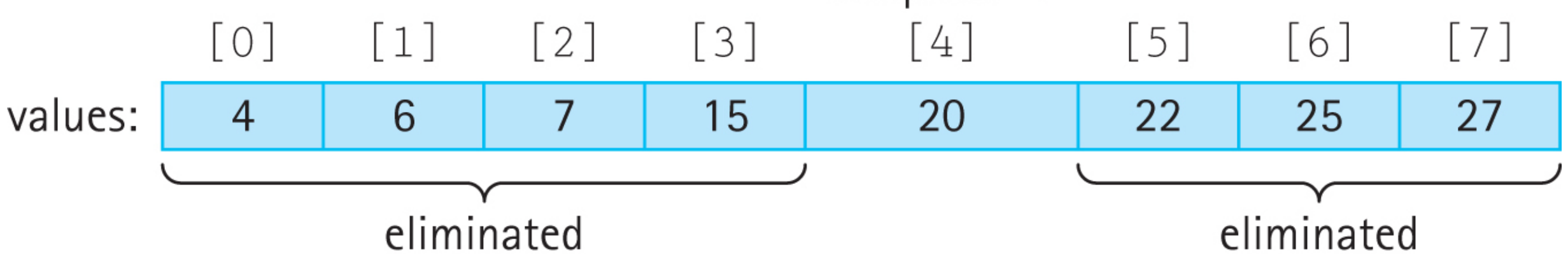
target: 20



Find

target: 20

first=4
last=4
midpoint=4



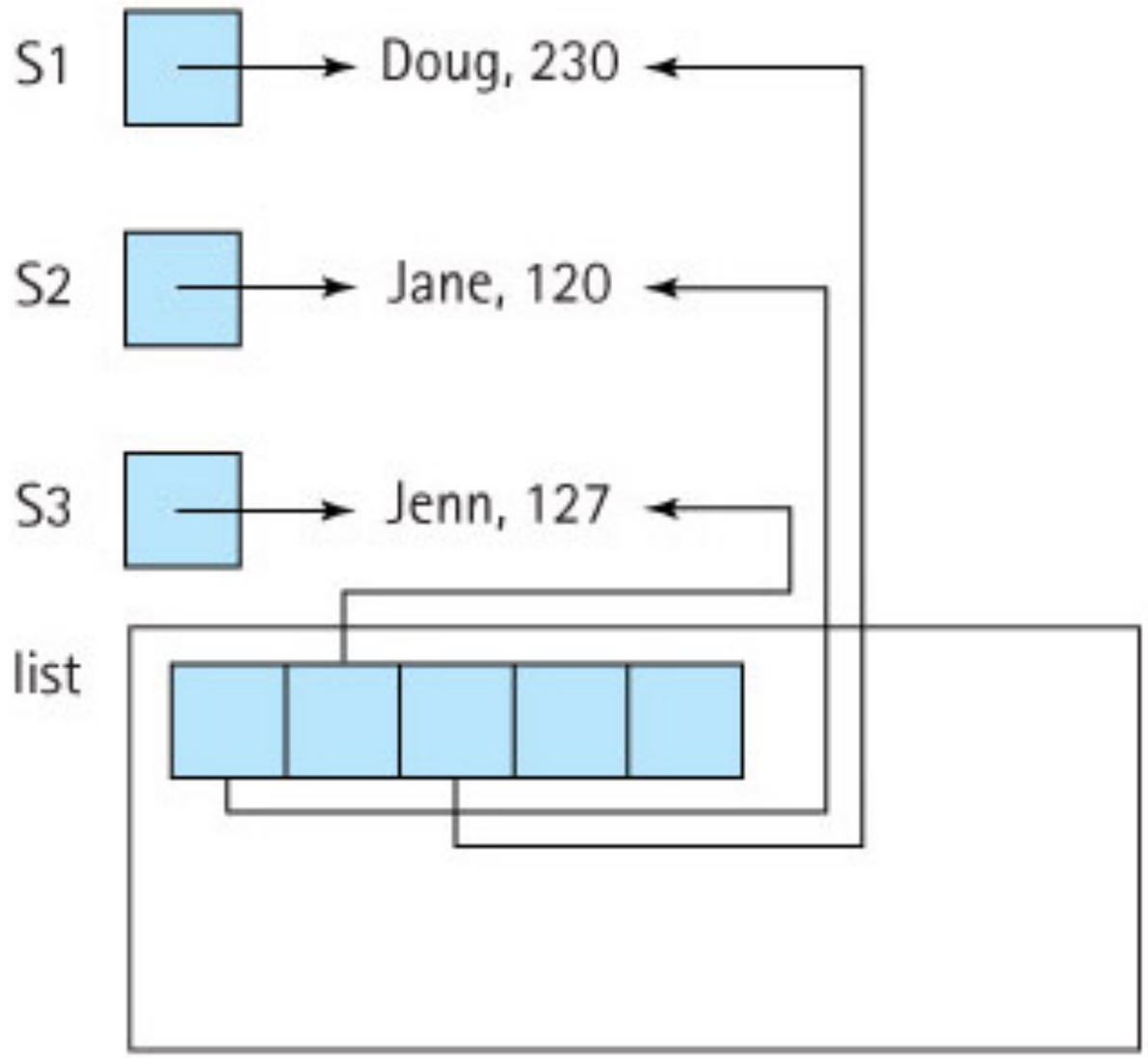
Find

```
protected void recFind(T target, int first, int last)
// Used by find.
{
  int result;         // result of the comparison
  if (first > last)
  {
    found = false;
    result = ((Comparable)target).compareTo(elements[location]);
    if (result > 0)
       location++;    // adjust location to indicate insert index
  }
  else
  {
    location = (first + last) / 2;
    result = ((Comparable)target).compareTo(elements[location]);
    if (result == 0)  // found target
      found = true;
    else
    if (result > 0)   // target too high
      recFind(target, location + 1, last);
    else               // target too low
      recFind(target, first, location - 1);
   }
}
```

Concerns

- We leave it to the programmer to ensure the objects stored are comparable
- Recall: how does Java pass parameters?
- Is it possible for a user to modify objects in the Collection after they're added?

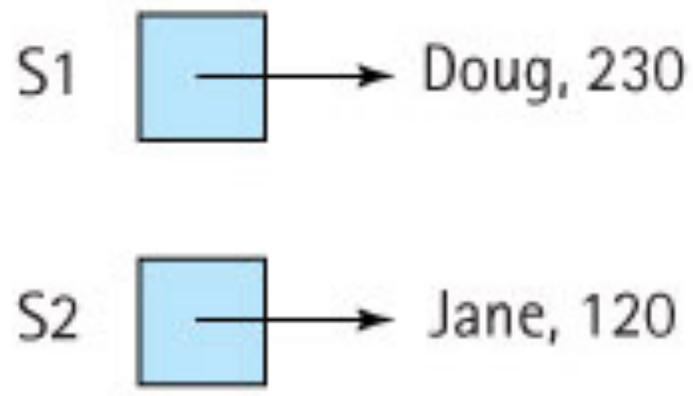
Maintaining and modifying references

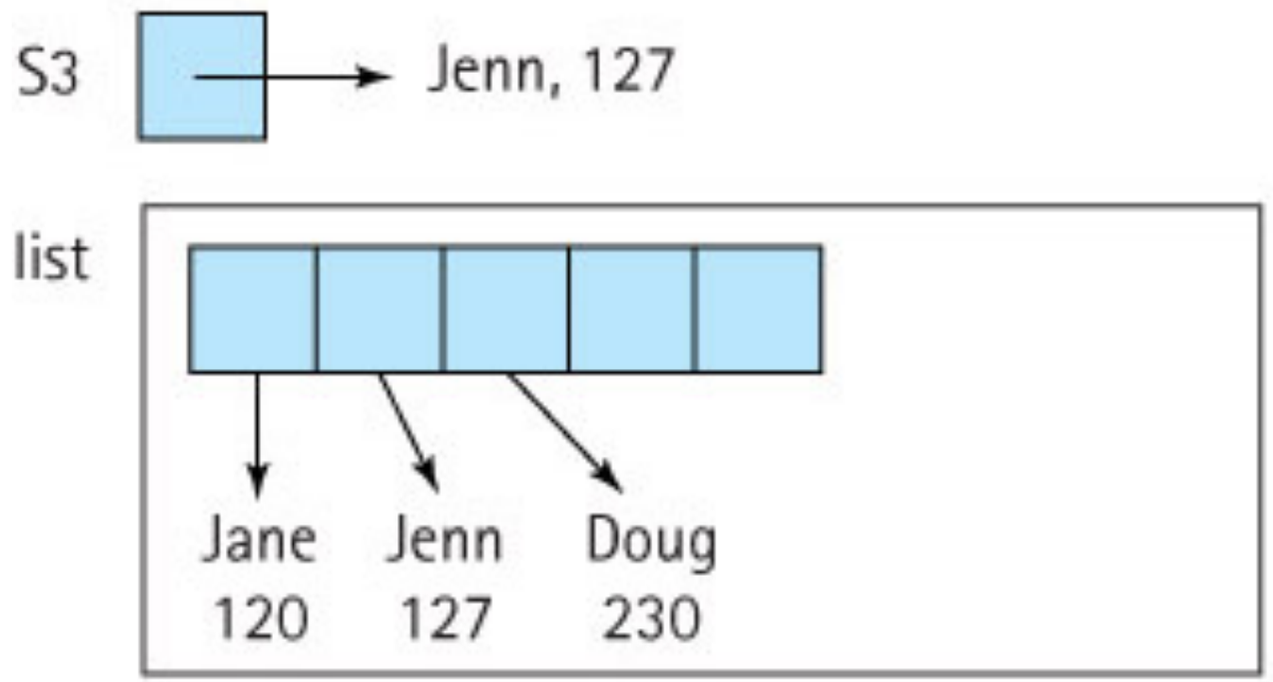


Forcing call-by-value for objects

- The clone() method allows objects to be copied
- Implemented as part of the Cloneable interface
- Ensures consistency after objects are stored

Protecting objects with cloning





Vocabulary Density Revisited

- Sorting the Collection caused a plethora of unexpected implementation issues
- Given that we are exchanging time in add() and remove() for time in contains(), was it worth it?
- Consider vocabulary density: every word requires a contains() call, but only some require add()

Vocabulary Density Results

Table 5.1 Results of Vocabulary Density Experiment[1]

Text	File Size	Results		`Array-Collection`	`Sorted-Array-Collection`
Shakespeare's 18th Sonnet	1 KB	words: unique: density:	114 83 1.37	20 msecs	23 msecs
Shakespeare's *Hamlet*	177 KB	words: unique: density:	32,247 4,790 6.73	236 msecs	128 msecs
Linux Word File	400 KB	words: unique: density:	45,404 45,371 1.00	9,100 msecs	182 msecs
Melville's *Moby-Dick*	1,227 KB	words: unique: density:	216,113 17,497 12.35	2,278 msecs or 2.3 seconds	382 msecs
The Complete Works of William Shakespeare	5,542 KB	words: unique: density:	900,271 26,961 33.39	9.7 seconds	1.2 seconds
Webster's Unabridged Dictionary	28,278 KB	words: unique: density:	4,669,130 206,981 22.56	4.7 minutes	9.5 seconds
11th Edition of the *Encyclopaedia Britannica*	291,644 KB	words: unique: density:	47,611,399 695,531 68.45	56.4 minutes	2.5 minutes
Mashup	608,274 KB	words: unique: density:	102,635,256 1,202,099 85.38	10 hours	7.2 minutes

Recap

- Comparison between objects must be defined individually for each object using equals()
- Objects are compared using "key" information, which may or may not be all of the stored data
- More refined comparison allowed between Comparable objects with the compareTo() method
- Comparison allows sorting, which speeds up contains() searches
- Object call-by-value with cloning vs call-by-reference

Next Time...

- Exam Review Session
  - Practice problems and bring questions!
- Check the course webpage for practice problems
- Peer Tutors
  - http://www.csc.villanova.edu/help/