# CS2040 Lab 6
# Priority Queue

# Lab 6 – Priority Queue

- Java provides a PriorityQueue class, which uses a binary heap
  - To create a new priority queue:
  - `PriorityQueue<Integer> pq = new PriorityQueue<Integer>();`
- Is a min heap by default; use Comparator.reverseOrder() to change it to a max heap (for elements that implement Comparable):
  - `PriorityQueue<Integer> pq = new PriorityQueue<Integer>(Comparator.reverseOrder());`
- PriorityQueue enforces some ordering on its elements. As a result, any object stored must either implement Comparable, or the constructor must be given a Comparator

# Lab 6 – PriorityQueue

| Method name | Description | Time |
|---|---|---|
| .add(YourClass element) | Adds *element* to the PriorityQueue | O(log n) |
| .clear() | Empties the PriorityQueue | O(n) |
| .contains(Object o) | Checks if *o* is in the PriorityQueue, based off the object's equals() method | **O(n)** |
| .peek() | Returns the top element of the PriorityQueue | O(1) |
| .poll() | Removes and returns the top element of the PriorityQueue | O(log n) |
| .remove(Object o) | Removes *o* if it is in the PriorityQueue, based off the object's equals() method | **O(n)** |
| .size() | Returns the number of elements in the list | O(1) |

# Lab 6 – Priority Queue

# PQ for thought

What if I want a PQ using binary heaps that can

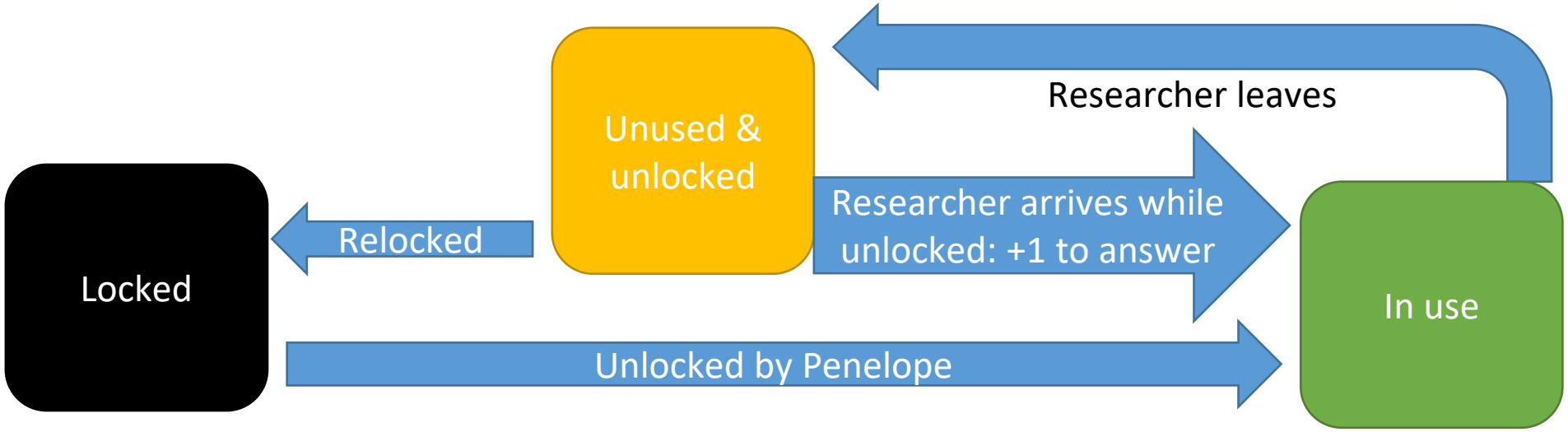
- Remove any element in O(log n)?
- Merge with another pq in O(n)?
- Remove the top and bottom element in O(log n)?
- Update the priority of an entry? (idea used later in semester)

(Not in syllabus) What if I want a PQ that can

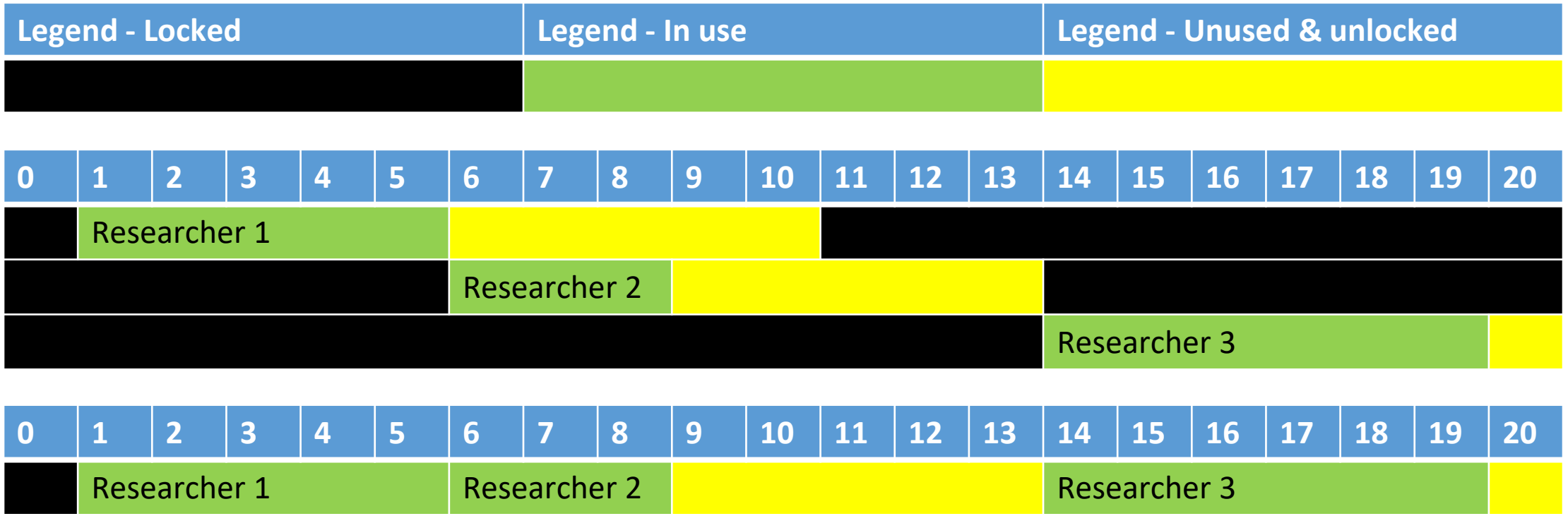
- Merge with another pq in O(log n)?
- Remove top and bottom element in O(log n), without using any HashMap / multiple data structures?

# One-Day Assignment 5 – Assigning Workstations

- Workstations can be in one of three different states:
  - Locked (so Penelope has to unlock this workstation)
  - In use (so this workstation cannot be assigned to a different researcher yet)
  - Unused, but unlocked (this workstation can be assigned to a researcher, so Penelope does not need to unlock a new workstation)



# One-Day Assignment 5 – Assigning Workstations

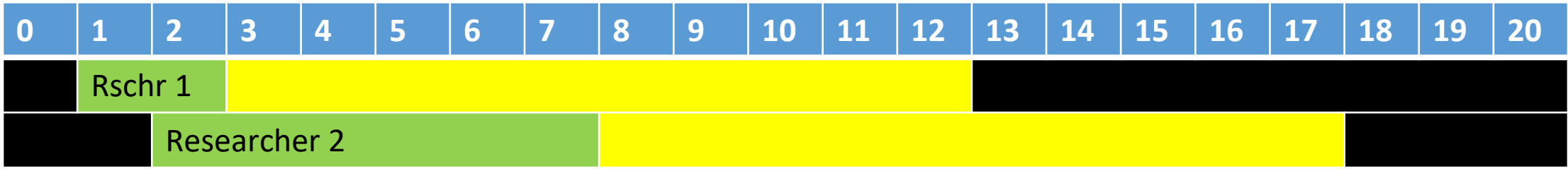



The above is an illustration of Sample Input 1
The first version requires Penelope to unlock a new workstation every time a new researcher enters
The second version requires Penelope to only unlock a workstation 1 time instead of 3 times, thereby saving **2** unlocks

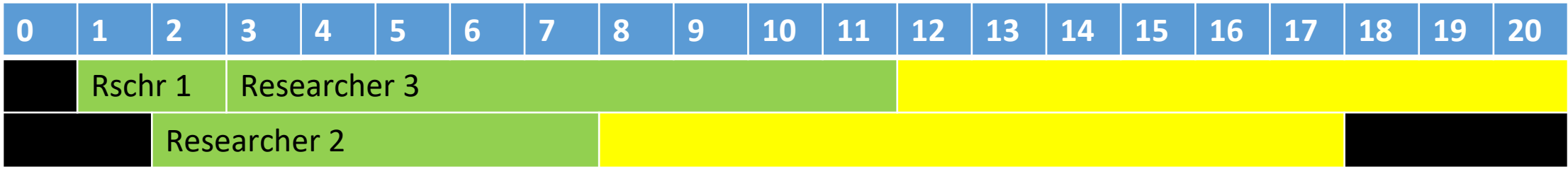
# Assigning Workstations – Sample Input 2


1 2 – First researcher, no choice but to unlock new WS

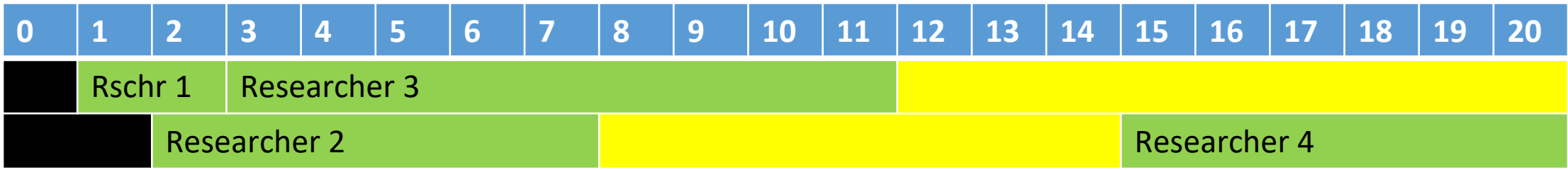

2 6 – First relocking WS (WS1) still in use at time 3, unlock new WS




3 9 – First relocking WS (WS1) is unused and unlocked at time 3, assign it




15 6 – First relocking WS (WS2) is unused and unlocked at time 15, assign it




17 7 – First relocking WS (WS1) is unused and unlocked at time 17, assign it