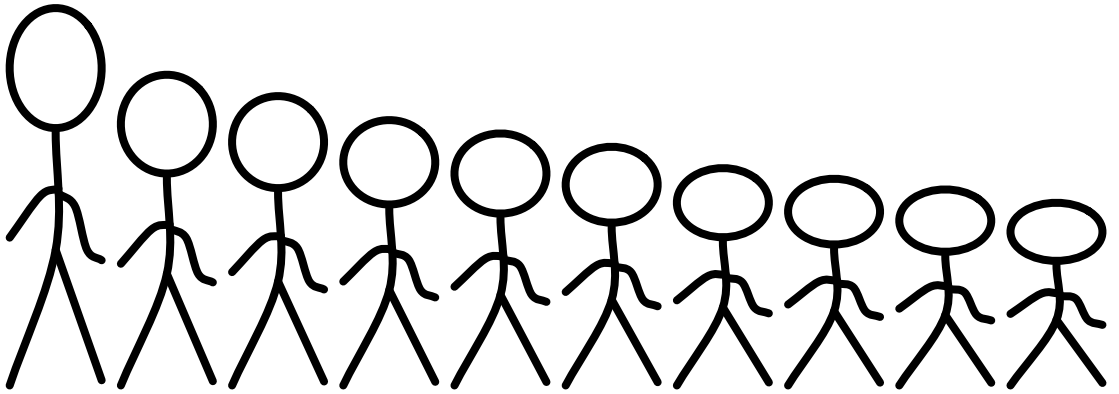
Priority Queues

In a **normal queue**, you put stuff in on one end, and remove from the other. Like waiting in line for whatever these people are doing. You start by going to the end of the line, then you wait until the people in front of you are taken care of. You leave the line (the queue) only when you are the person in front.

back of the line

front of the line

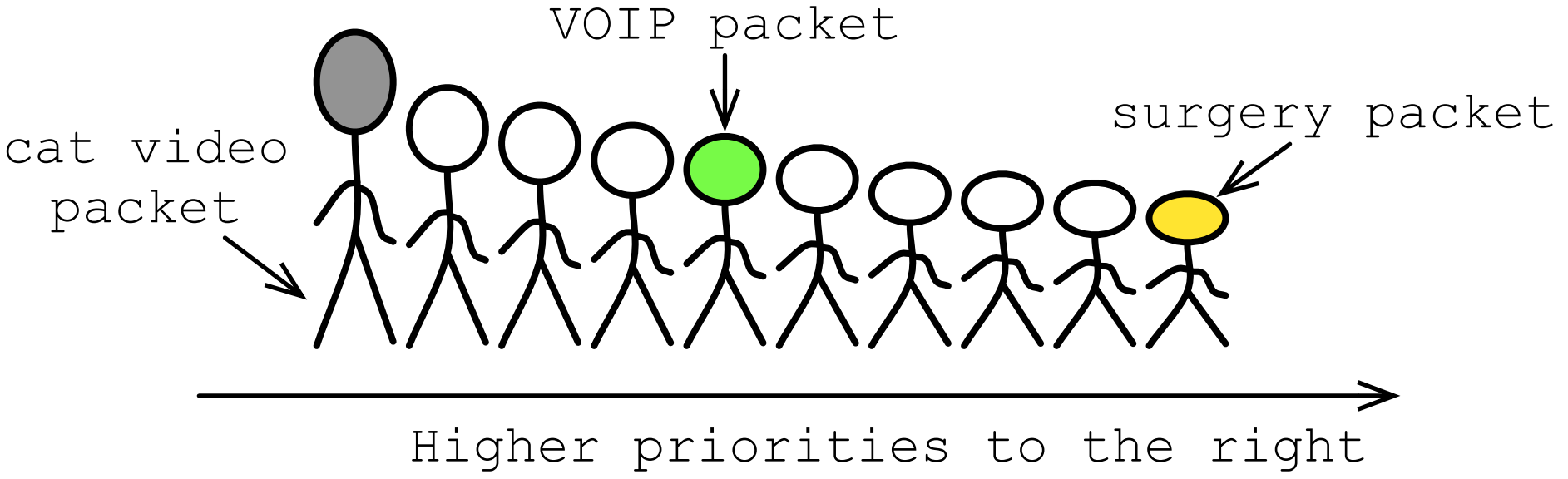
A *Priority* queue is an abstract structure where the item with the highest priority is chosen next.

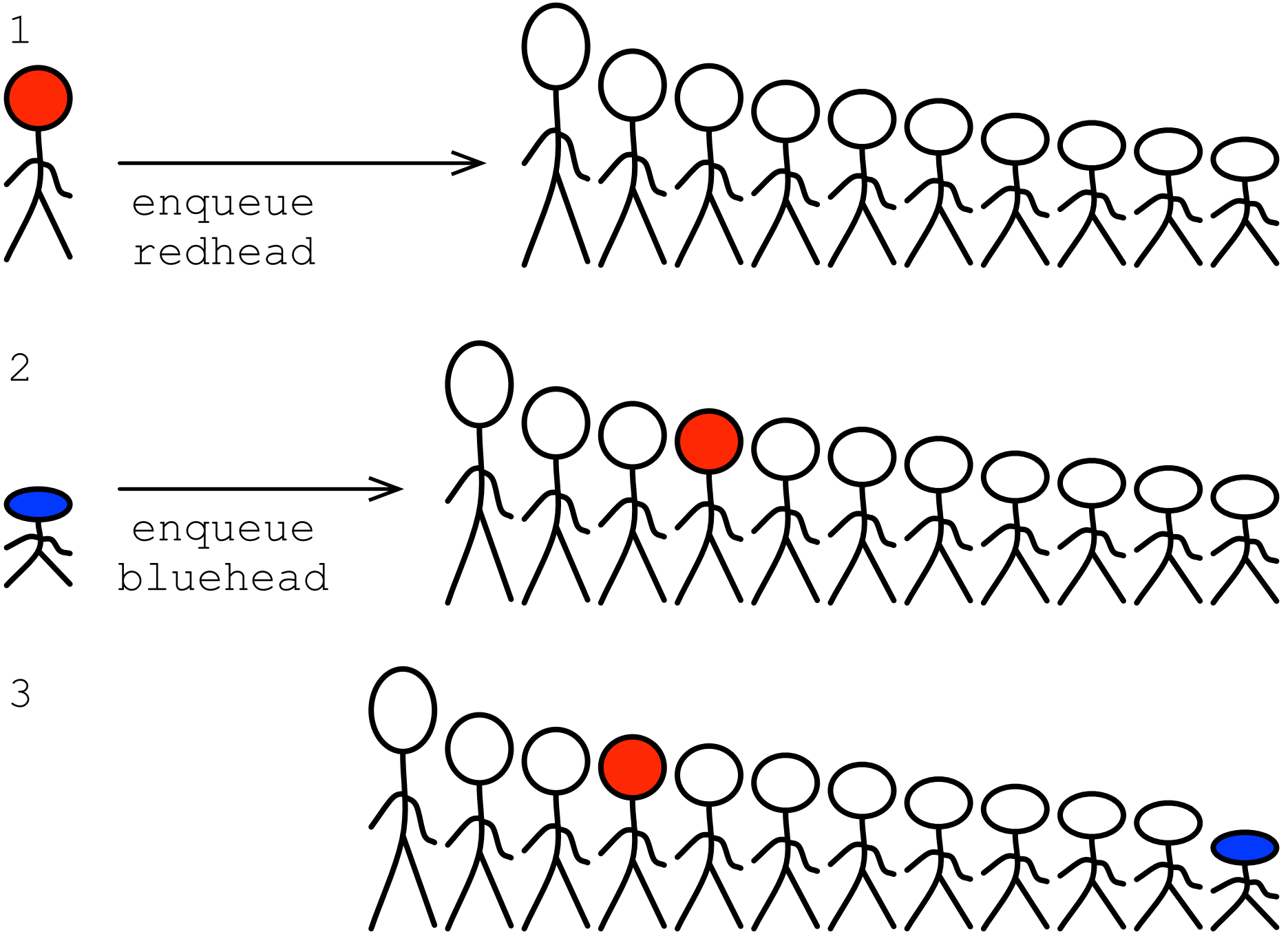


Here is a very unfair line at the ticket booth. It is a priority queue that is based on the person's height. Short people start out closer to the front of the line than taller people.

Why we do this

A priority queue is often used to discriminate between important actions (e.g. packets for tele-surgery) and relatively less important packets (e.g. awesome cat videos). Deal with the important stuff before the cat videos.




1
enqueue
redhead
2
enqueue
bluehead
3

You don't have to implement this using a list.

You could use:

* **An unsorted list**, and scan through the entire thing when you want to find the next item.

* **A sorted list**, where you insert items in the right place.

* **A heap**, where items are stored in a heap data structure (binary trees and arrays can be used), and the one on top is always the next to be used.

Remember the lecture on heapsort? You could implement this assignment using heapsort. Or you could implement the double-ended queue with a special insertion strategy that uses priority.

Abstract Data Type vs Concrete Data Structures

Since you can implement a priority queue using a bunch of different strategies, we call it an *abstract data type*. All we care about is the interface that it exposes: we put stuff in with a given priority, and we take a thing out that has the highest priority in the structure.

The red-black tree tree we looked at a few weeks ago is a concrete data structure, because it specifies *how* the thing works.

A priority queue only specifies the interface, but not how we implement that interface.

Priority Queue Interface

insert(pq, thing, priority)

Insert the given *thing* into the priority queue *pq* with the specified priority.

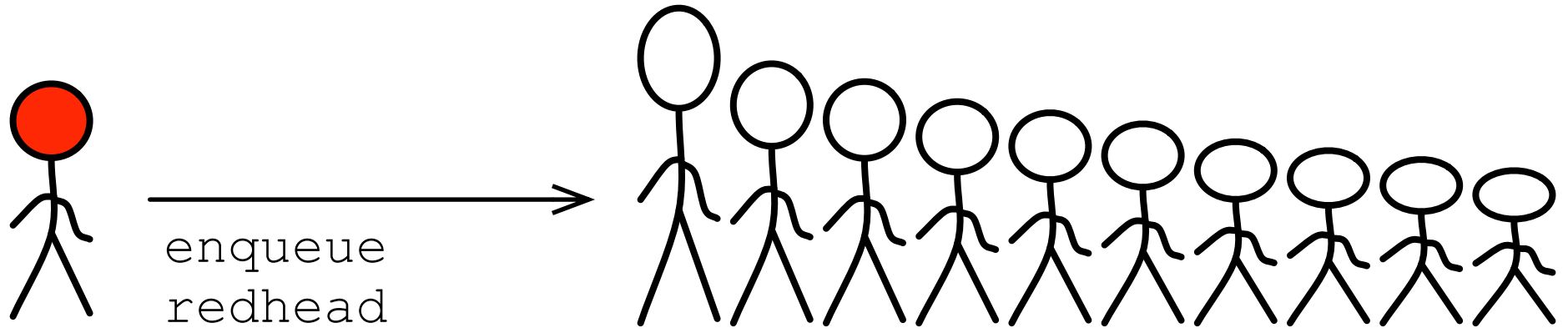
remove(pq)

Remove and return the highest priority item in the priority queue *pq*.

peek(pq)

Like 'remove' but do not remove the item.

Priority Queue: Insert




```
void insert(pq* &queue, string &data,
            float priority);
```

Let's say we have a linked list with a modified insertion function.

Do we need to keep the priority value around?

What's the computational complexity of inserting a new item this way? (hint: it depends how you implement it...)

What about a tie?

Good question!

Depends on what you're doing.

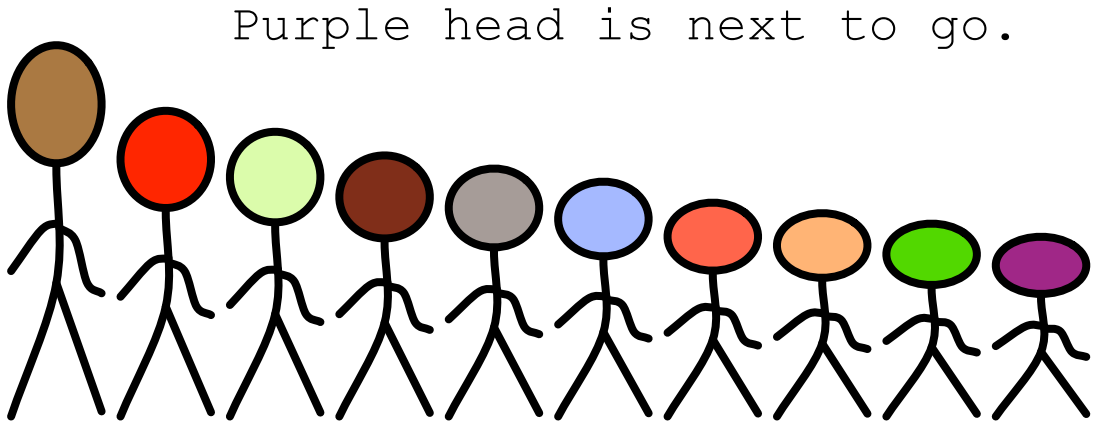
Option 1: Do nothing. Don't insert.
Option 2: Replace the current item at that priority level.
Option 3: Put the new item at the end of the pack of items with the same priority.
Option 4: Put the new item somewhere among the items with the same priority.

For our assignment,

we will use option 3.

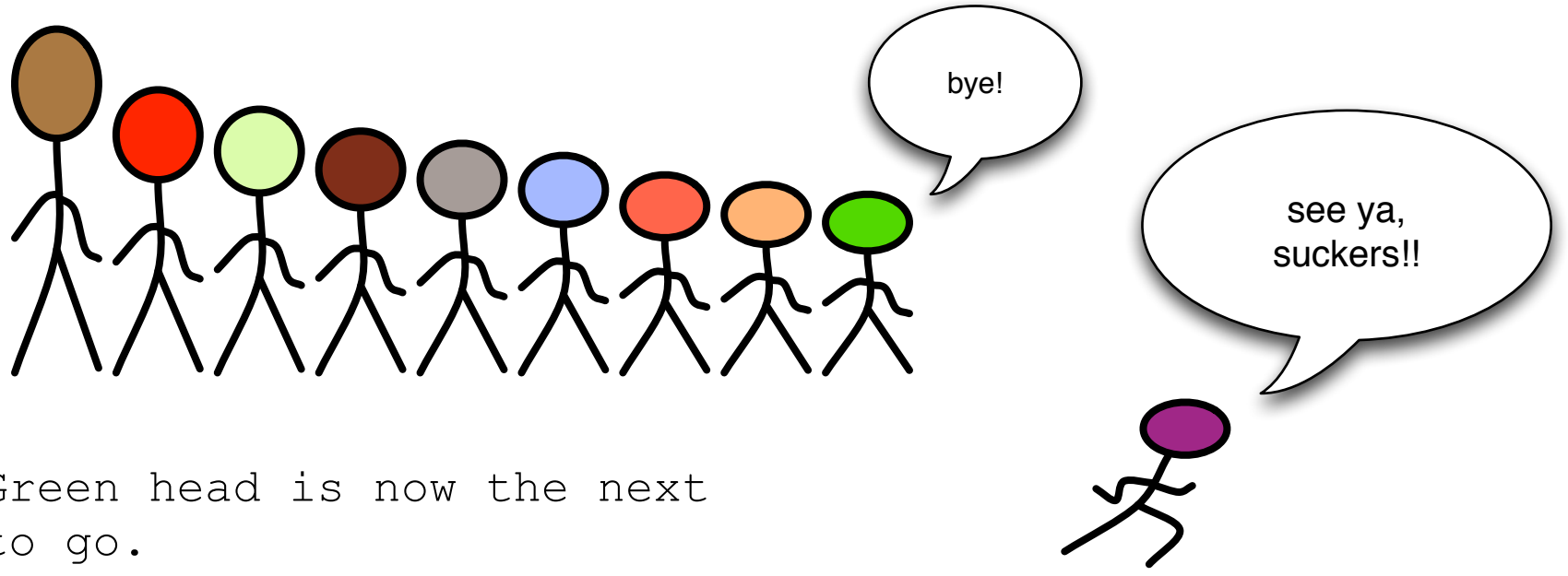
```
string remove(pq* &queue);
```

Removing can be constant time (e.g. if all you need to do is move a pointer), or O(log n) if you need to clean up a heap after a remove, or worse, if you're using a silly implementing data structure.



```
string remove(pq* &queue);
```

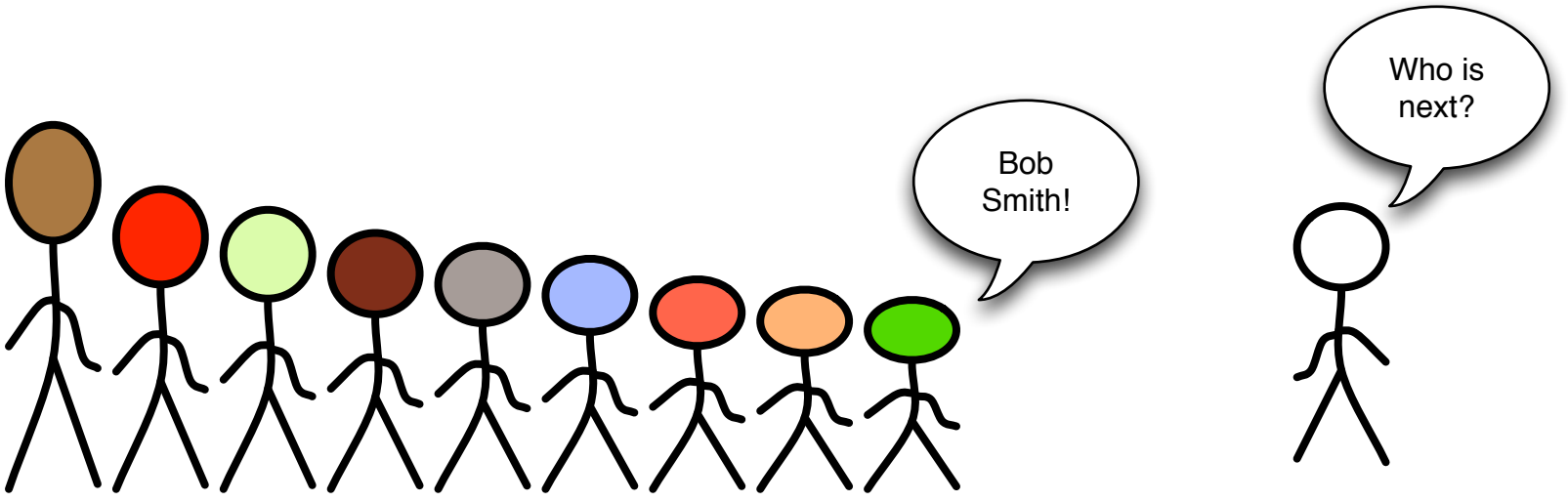
Removing can be constant time (e.g. if all you need to do is move a pointer), or O(log_n) if you need to clean up a heap after a remove, or worse, if you're using a silly implementing data structure.



Green head is now the next to go.

```
string peek(pq* &queue);
```

Peeking is easy. Just return the value that is currently at the highest priority position. Don't modify the underlying data structure.



This is like asking the high priority item to report its name without leaving the line.

The operations are straightforward.

The head trip with this assignment is that you will need to decide how you want to implement it.

This means: do you use a linked list? a heap? the vector class in the standard template library? something else?

However you do it, you'll need to specify the details in your own header file.

You will edit pq.cpp *and* pq_struct.h. You will turn in both files to retrograde.