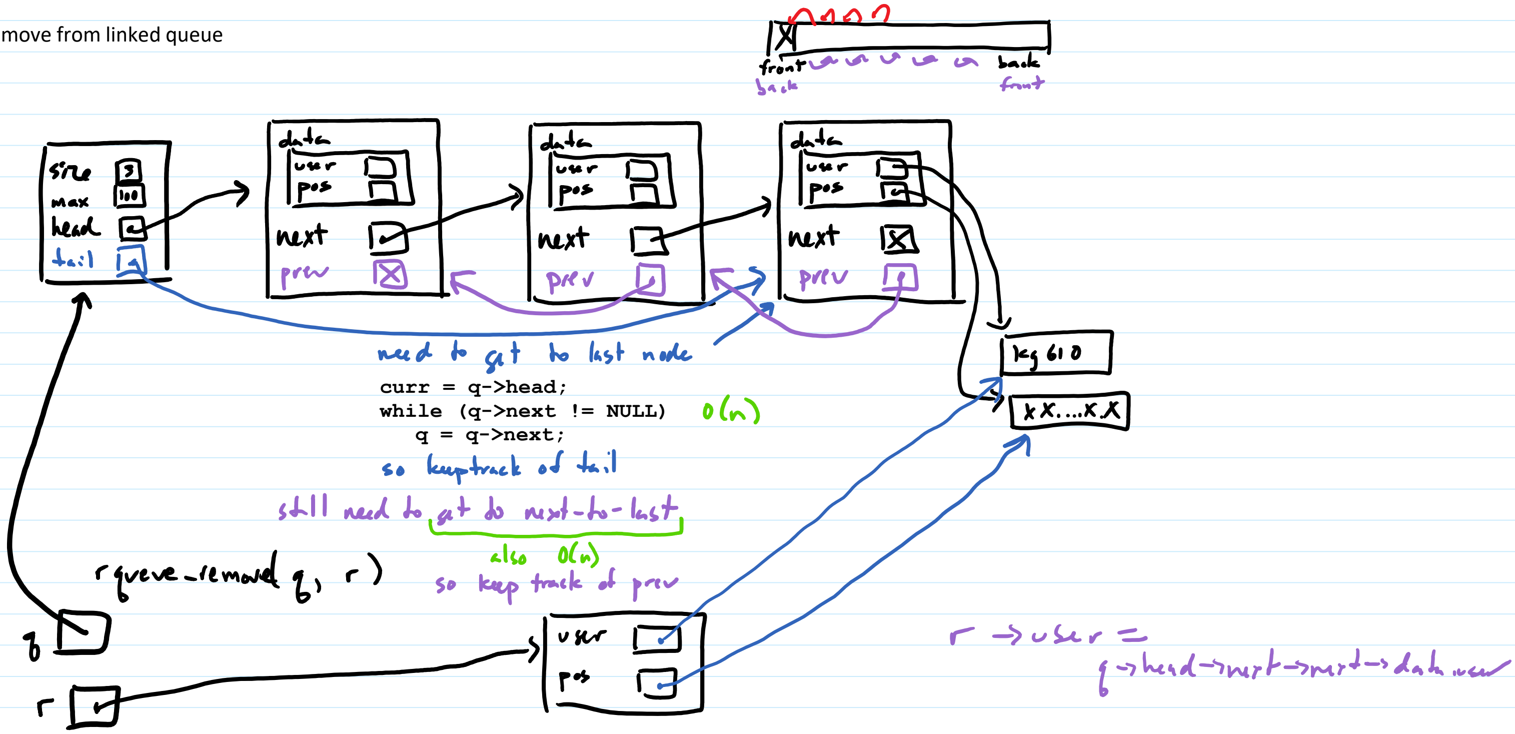
Remove from linked queue



need to get to last node

```
curr = q->head;
while (q->next != NULL)
   q = q->next;
```

O(n)

so keep track of tail

still need to get to next-to-last

also O(n)

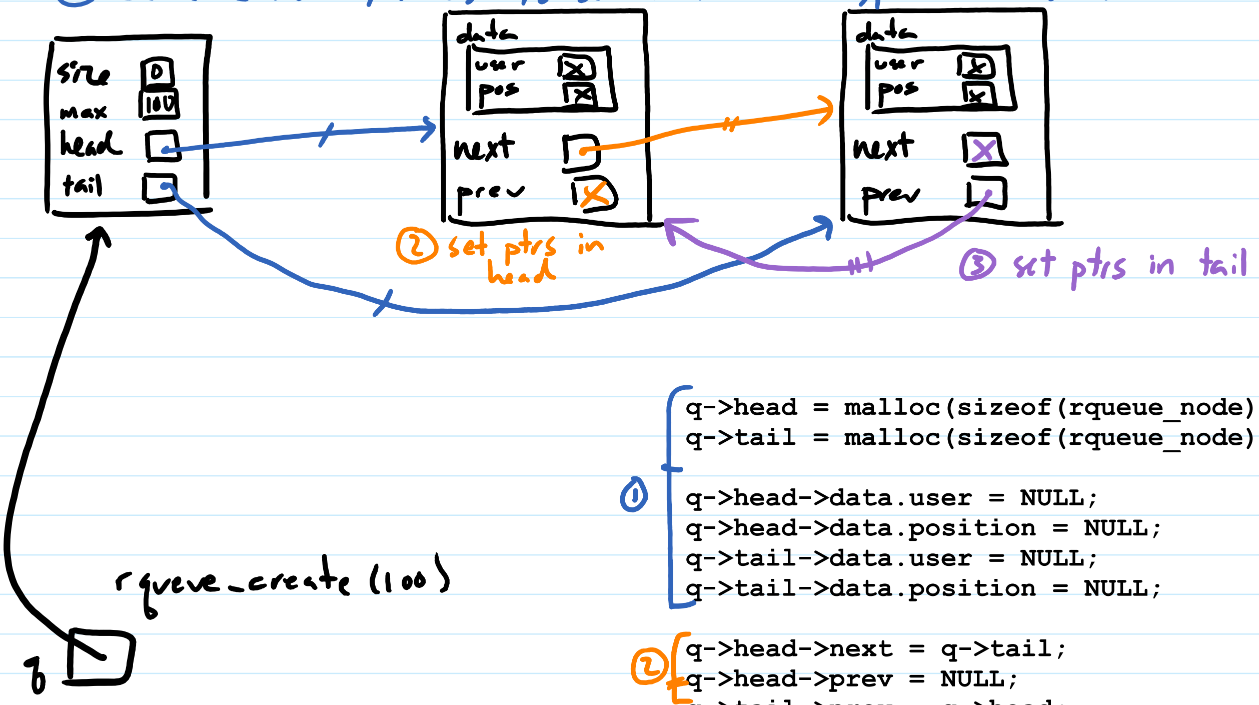
so keep track of prev

rqueue_remove(q, r)

r->user =
q->head->next->next->data.user

Create with dummy

① create 2 dummy nodes w/o data to reduce special cases (at cost of bookkeeping)



①

```
q->head = malloc(sizeof(rqueue_node));
q->tail = malloc(sizeof(rqueue_node));

q->head->data.user = NULL;
q->head->data.position = NULL;
q->tail->data.user = NULL;
q->tail->data.position = NULL;
```

②

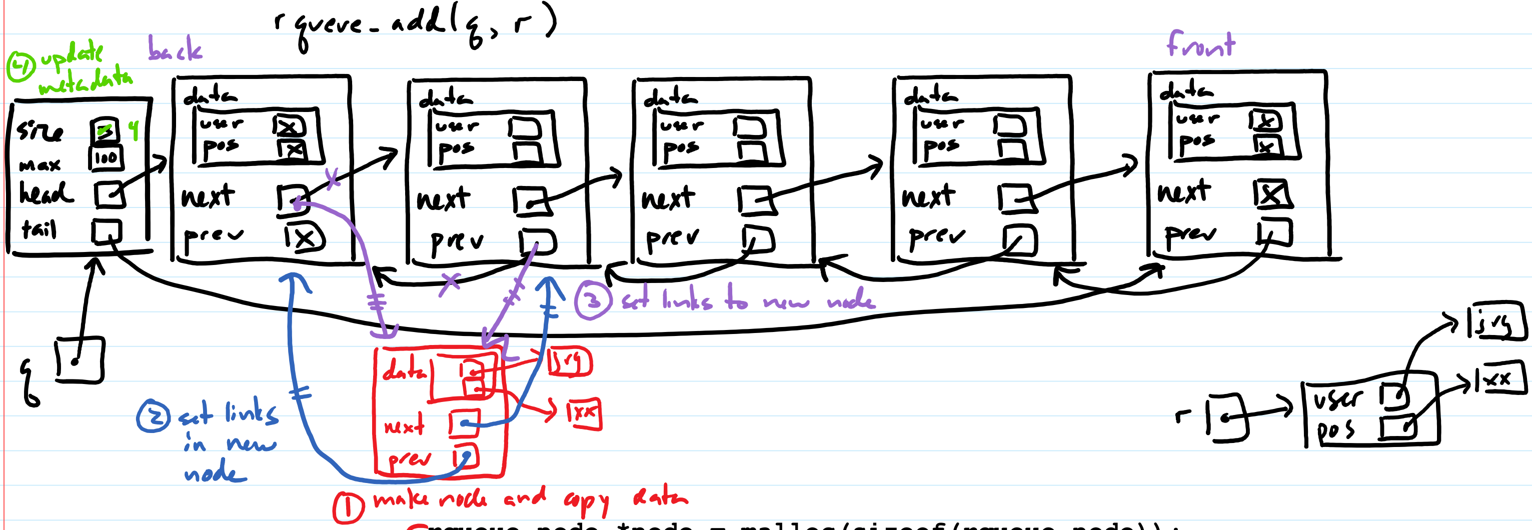
```
q->head->next = q->tail;
q->head->prev = NULL;
```

③

```
q->tail->prev = q->head;
q->tail->next = NULL;
```

From <http://zoo.cs.yale.edu/classes/cs223/f2018/Examples/Queue/request_queue_linked.c>

Add to linked queue



①

```
rqueue_node *node = malloc(sizeof(rqueue_node));
char *copy_user = malloc(sizeof(char) * (strlen(r->user) + 1));
char *copy_pos = malloc(sizeof(char) * (strlen(r->position) + 1));
strcpy(copy_user, r->user);
strcpy(copy_pos, r->position);
node->data.user = copy_user;
node->data.position = copy_pos;
```

②

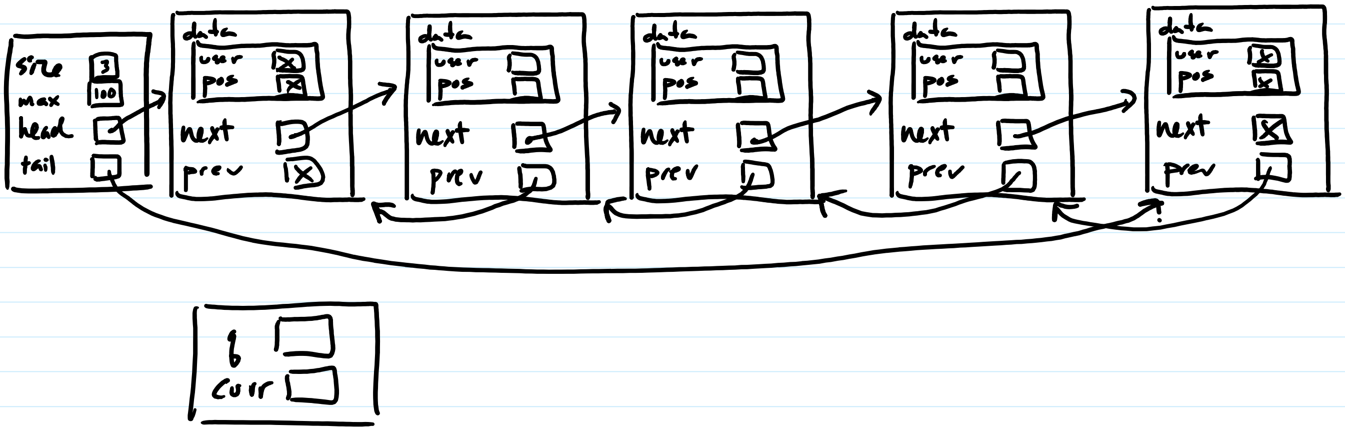
```
node->next = q->head->next;
node->prev = q->head;
```

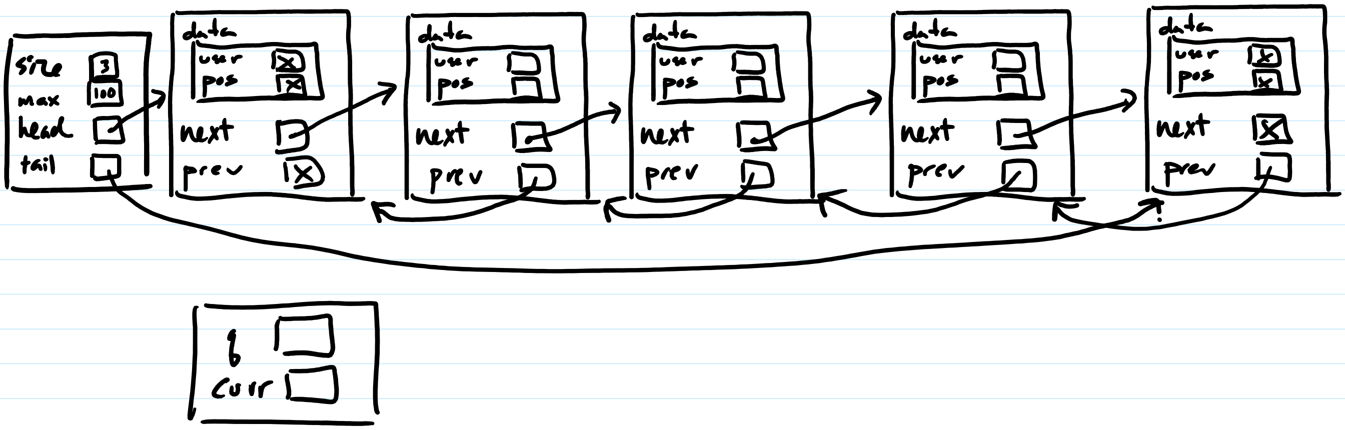
③

```
q->head->next->prev = node;
q->head->next = node;
```

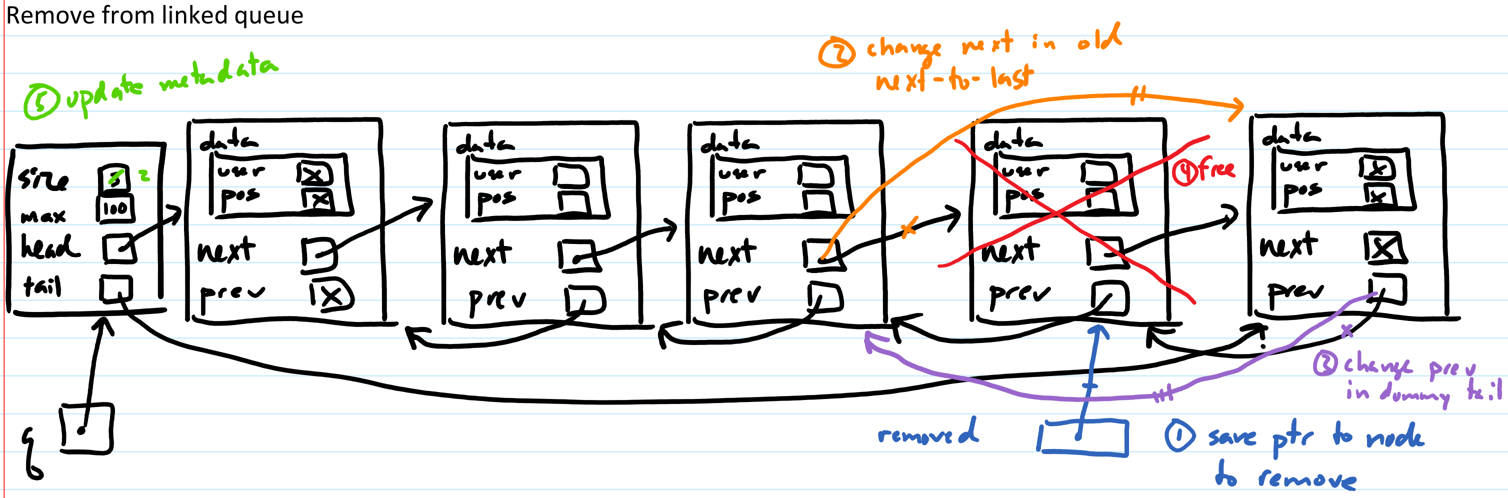
④

```
q->size++;
```


size
3
max
100
head
tail
data
user
pos
next
prev
q
curr


size
3
max
100
head
tail
data
user
pos
next
prev
q
curr

Remove from linked queue

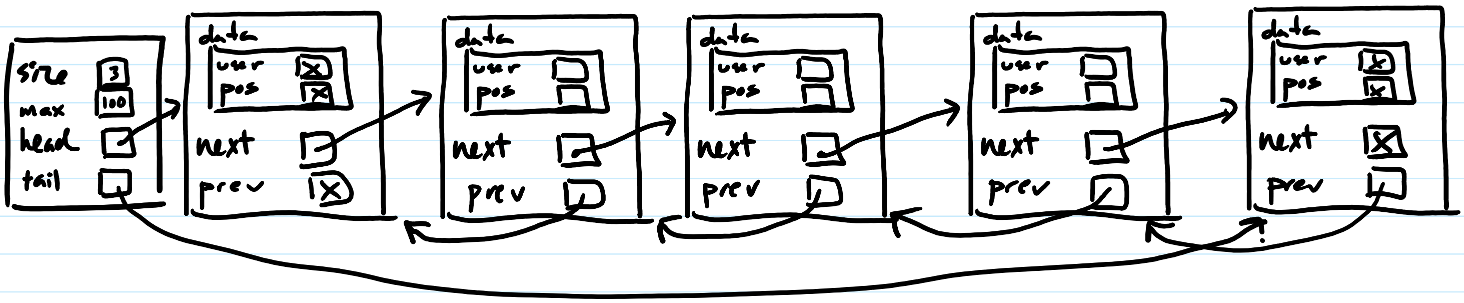

⑤ update metadata
② change next in old next-to-last
size
max
head
tail
3
2
100
data
user
pos
next
prev
④ free
③ change prev in dummy tail
q
removed
① save ptr to node to remove

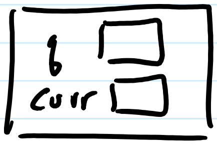
① `rqueue_node *removed = q->tail->prev;`

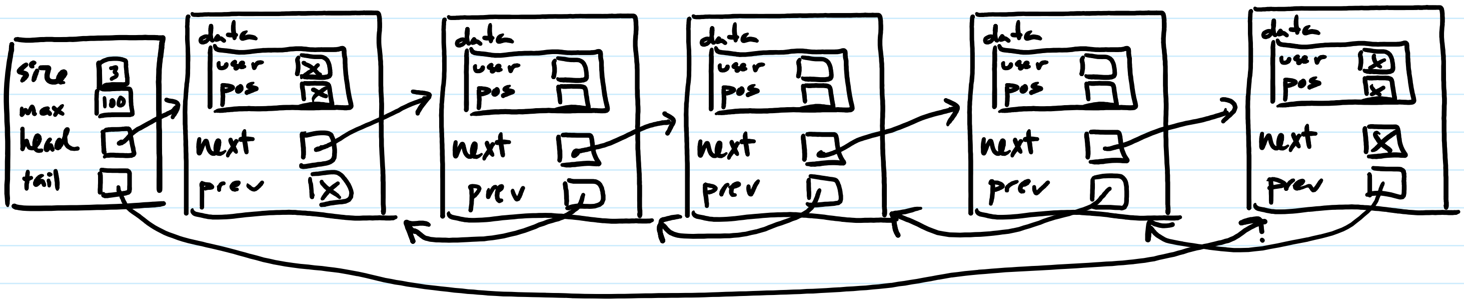
② `removed->prev->next = q->tail;`
③ `q->tail->prev = removed->prev;`

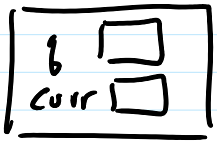
④ `free(removed);`

⑤ `q->size--;`


size
3
max
100
head
tail
data
user
pos
next
prev
data
user
pos
next
prev
data
user
pos
next
prev
data
user
pos
next
prev
data
user
pos
next
prev


8
curr

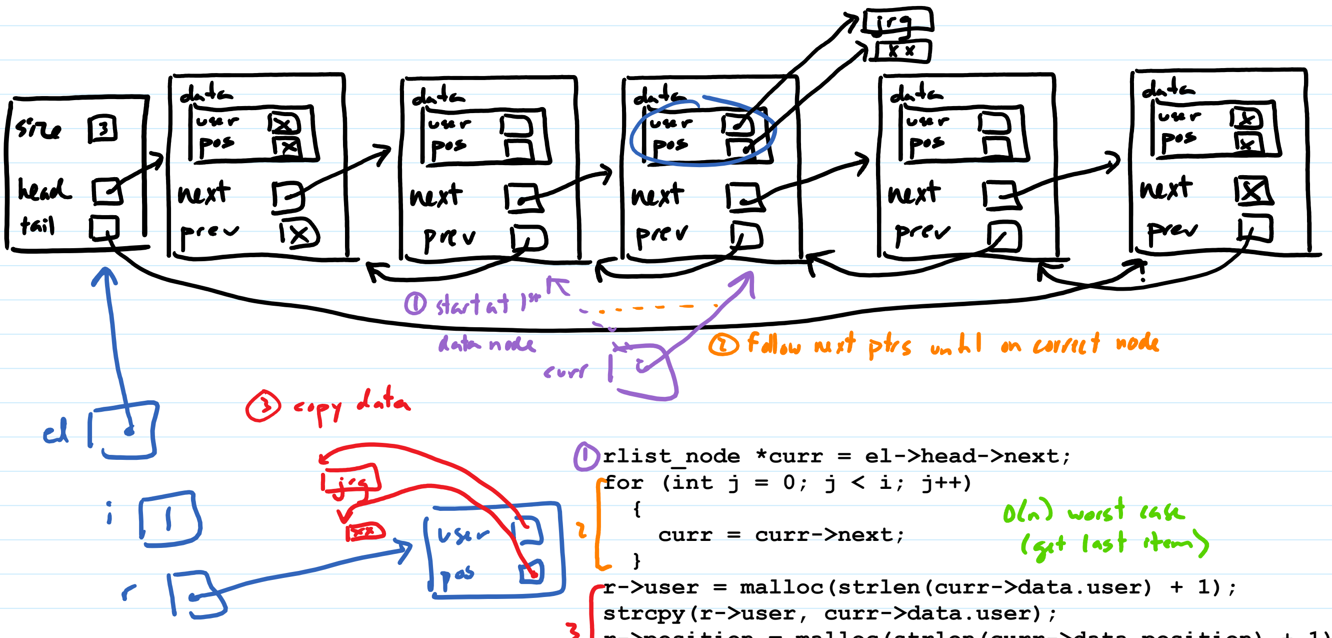

size
3
max
100
head
tail
data
user
pos
next
prev
data
user
pos
next
prev
data
user
pos
next
prev
data
user
pos
next
prev
data
user
pos
next
prev


b
curr

Iterators

rlist_get(el, 1, &r)

index of item to get

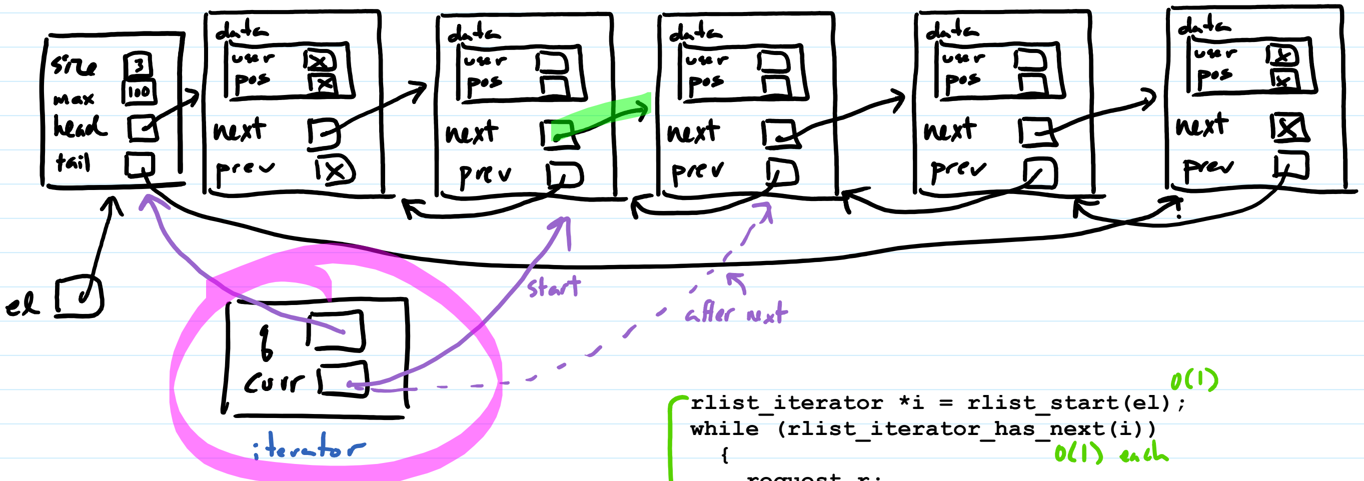


```
rlist_node *curr = el->head->next;
for (int j = 0; j < i; j++)
  {
    curr = curr->next;
  }
r->user = malloc(strlen(curr->data.user) + 1);
strcpy(r->user, curr->data.user);
r->position = malloc(strlen(curr->data.position) + 1);
strcpy(r->position, curr->data.position);
```

O(n) worst case (get last item)

```
for (int i = 0; i < rlist_size(el); i++)
  {
    request r;
    list_get(el, i, &r); // tot iters inside get = 0+1+2+ ... +n-1 = O(n^2)
    printf("%s %s\n", r.user, r.position);
    free(r.user);
    free(r.position);
  }
```

would be 1+1+...+1 = O(n) for array-based list



(a new opaque struct) O(n) total for linked

```
rlist_iterator *i = rlist_start(el);
while (rlist_iterator_has_next(i))
  {
    request r;
    rlist_iterator_get(i, &r);
    printf("%s %s\n", r.user, r.position);
    free(r.user);
    free(r.position);
```

O(1)

O(1) each

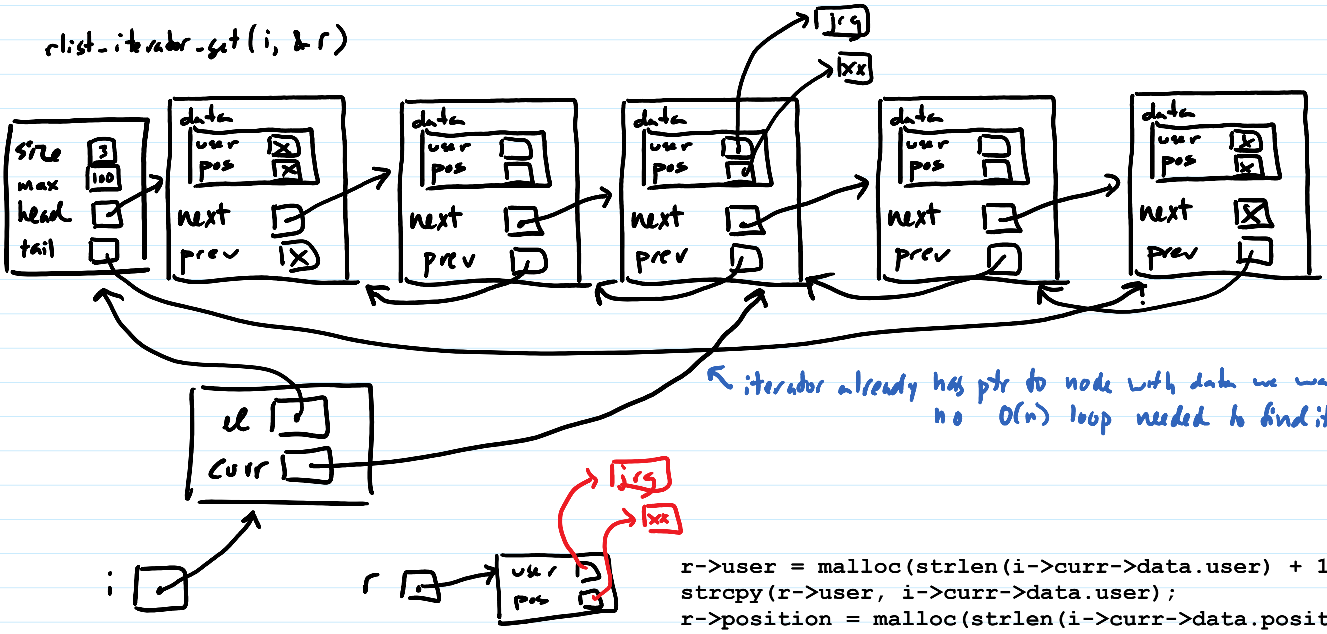
O(1) each

(a new opaque struct) O(n) total

for linked or array

```
    rlist_iterator_get(i, &r); O(1) each
    printf("%s %s\n", r.user, r.position);
    free(r.user);
    free(r.position);
    rqueue_iterator_next(i); O(1) each
  }
rqueue_iterator_destroy(i);
```

rlist_iterator_get(i, &r)

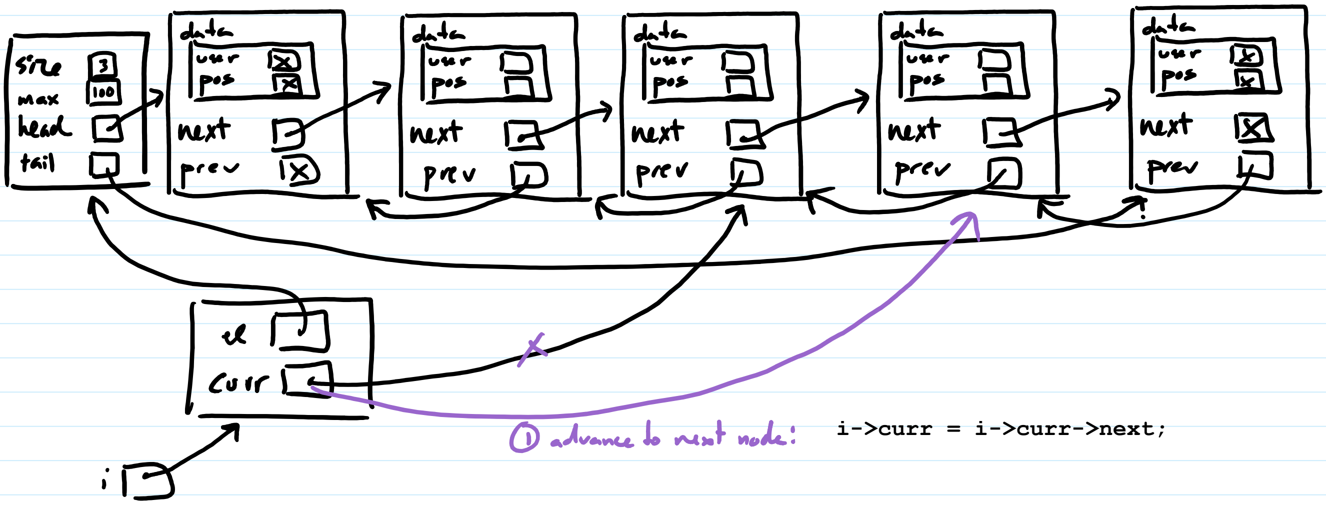


iterator already has ptr to node with data we want
no O(n) loop needed to find it!

```
r->user = malloc(strlen(i->curr->data.user) + 1);
strcpy(r->user, i->curr->data.user);
r->position = malloc(strlen(i->curr->data.position) + 1);
strcpy(r->position, i->curr->data.position);
```

O(1) [aside from time to copy data, which is not a fxn of size of list]

rlist_iterator_next(i)



① advance to next node: `i->curr = i->curr->next;`