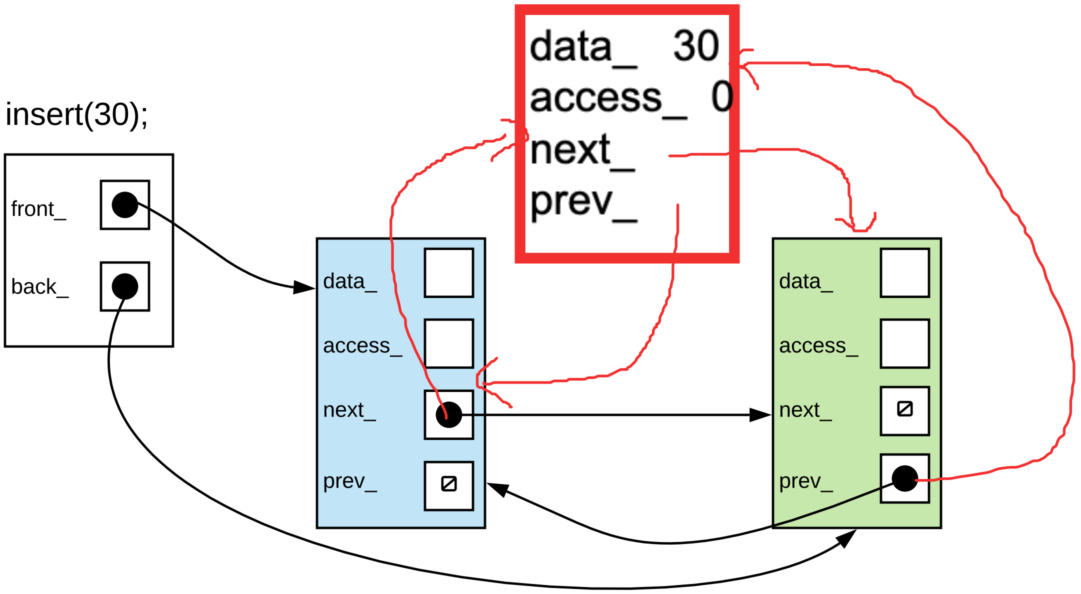
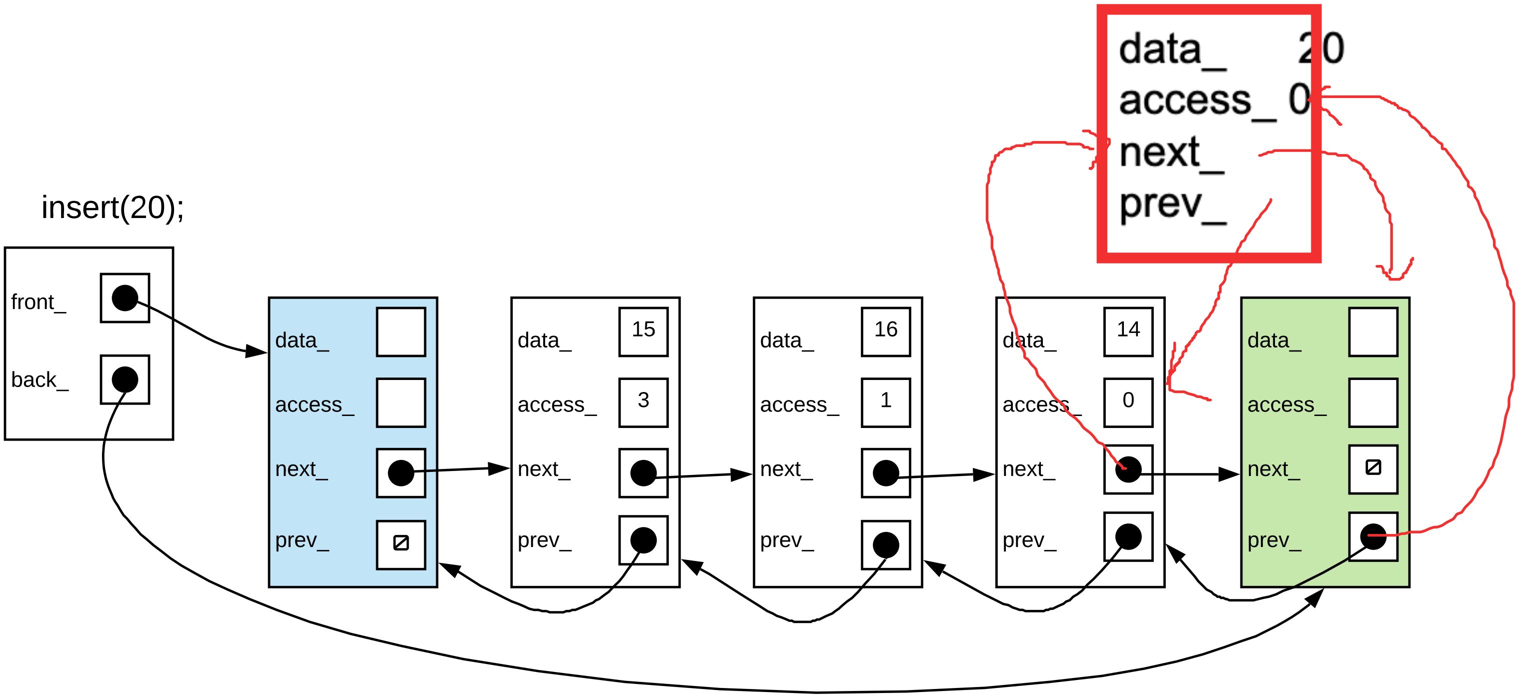
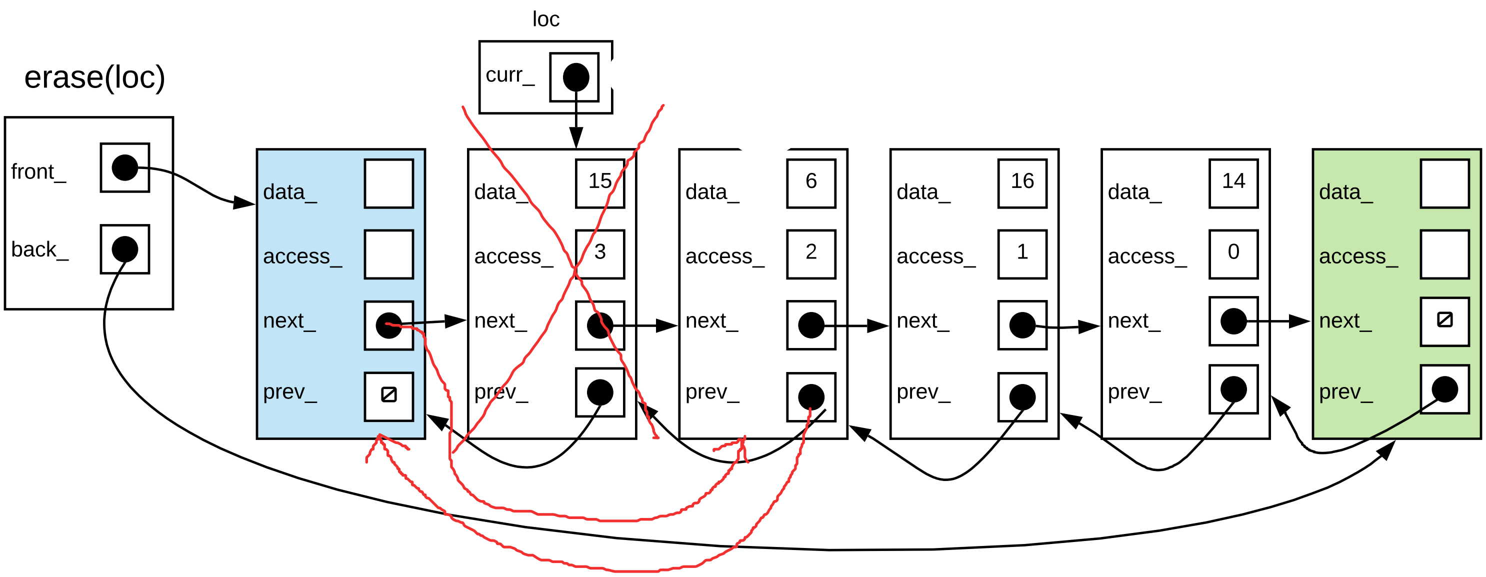
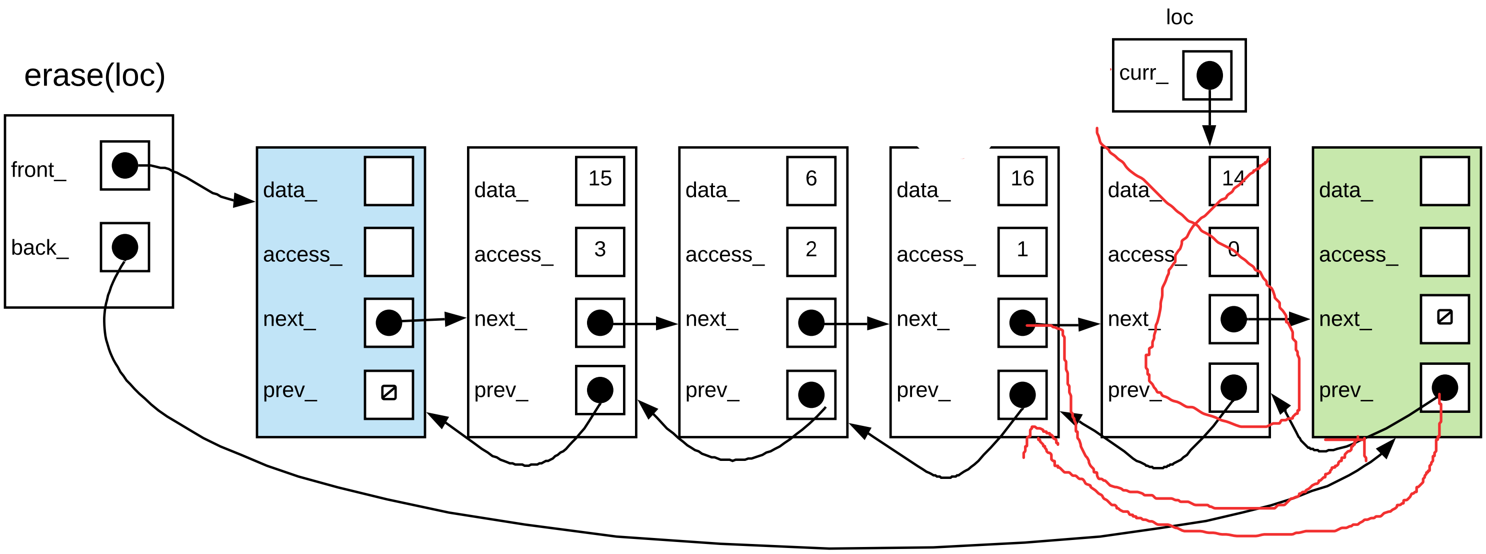

insert(30);
data_ 30
access_ 0
next_
prev_
front_
back_
data_
access_
next_
prev_
data_
access_
next_
prev_

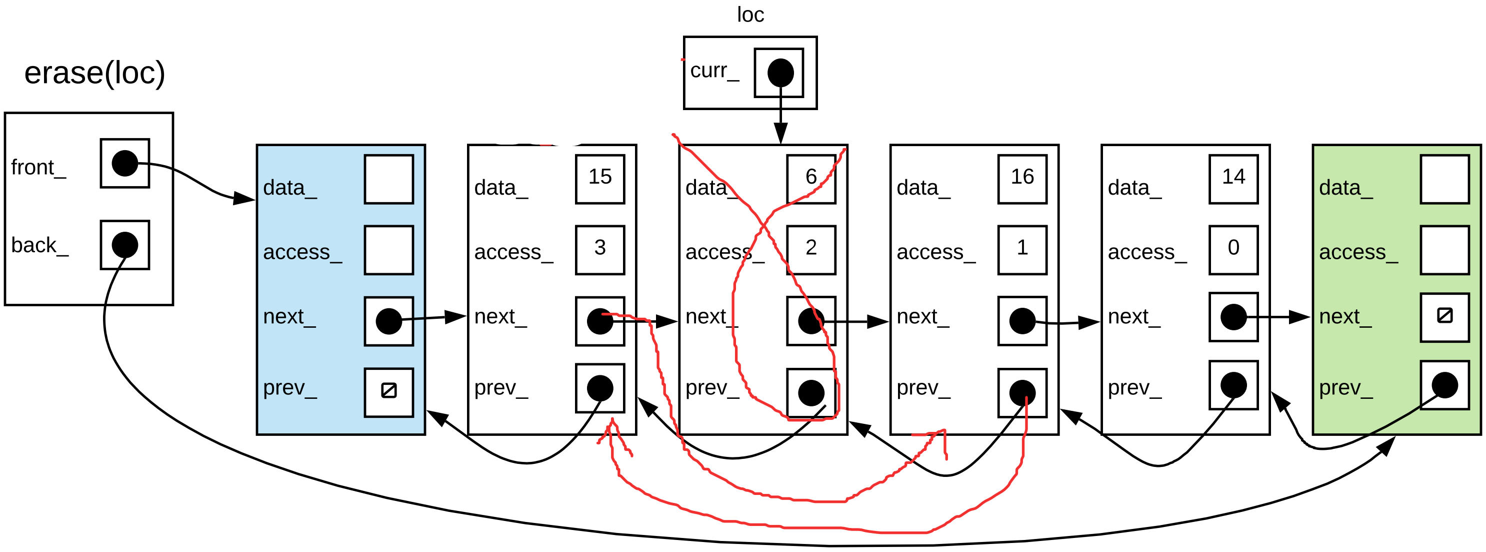

insert(20);
data_ 20
access_ 0
next_
prev_
front_
back_
data_
access_
next_
prev_
data_ 15
access_ 3
next_
prev_
data_ 16
access_ 1
next_
prev_
data_ 14
access_ 0
next_
prev_
data_
access_
next_
prev_

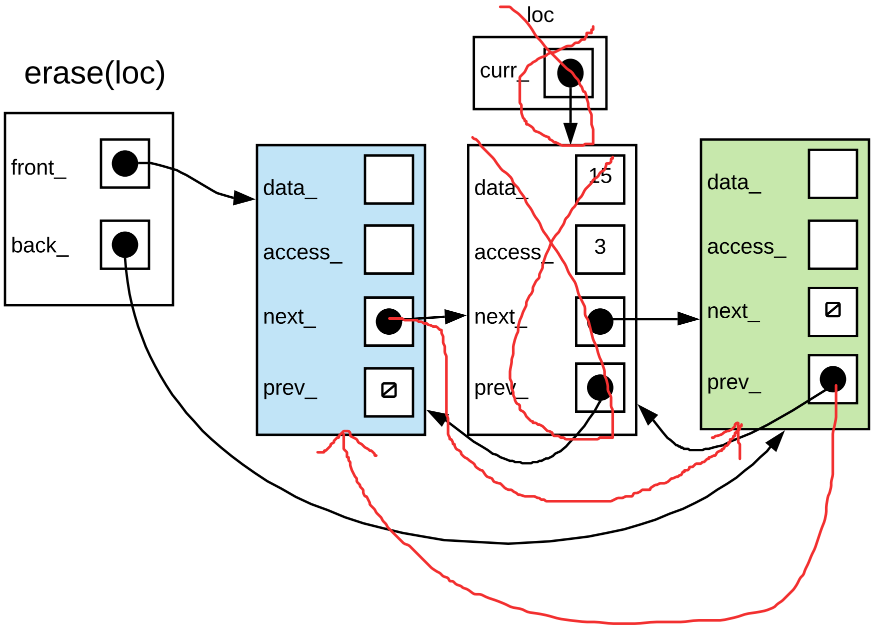
erase(loc)



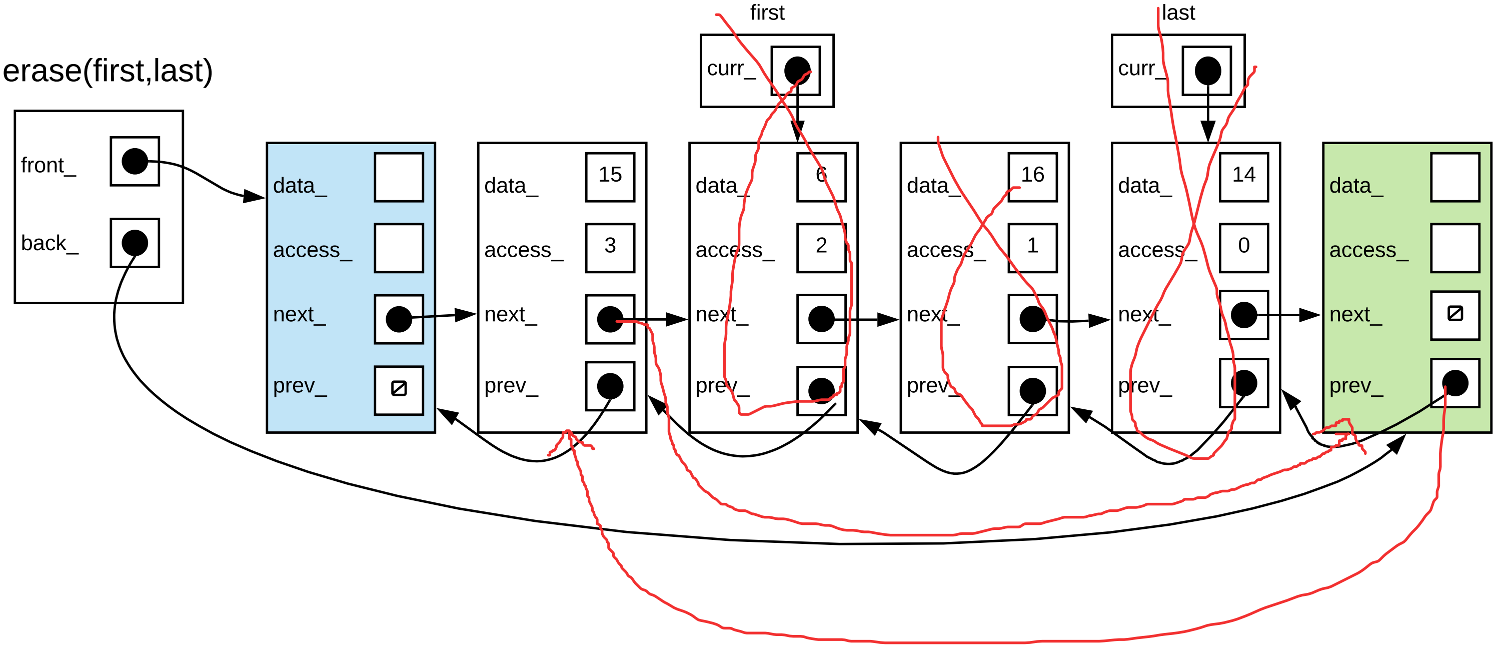
erase(loc)



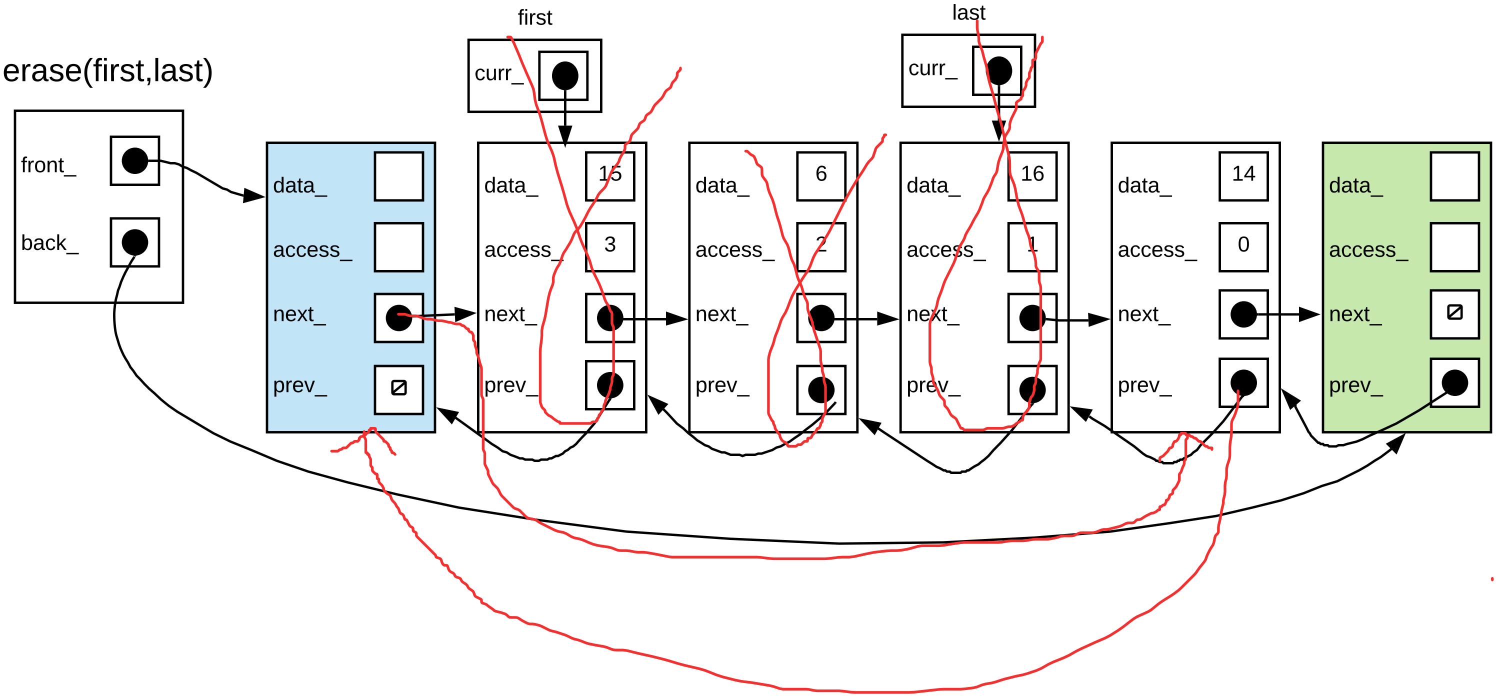

erase(loc)
loc
curr_
front_
back_
data_
access_
next_
prev_
15
3
6
2
16
1
14
0


erase(loc)
loc
curr_
front_
back_
data_
access_
next_
prev_
15
3

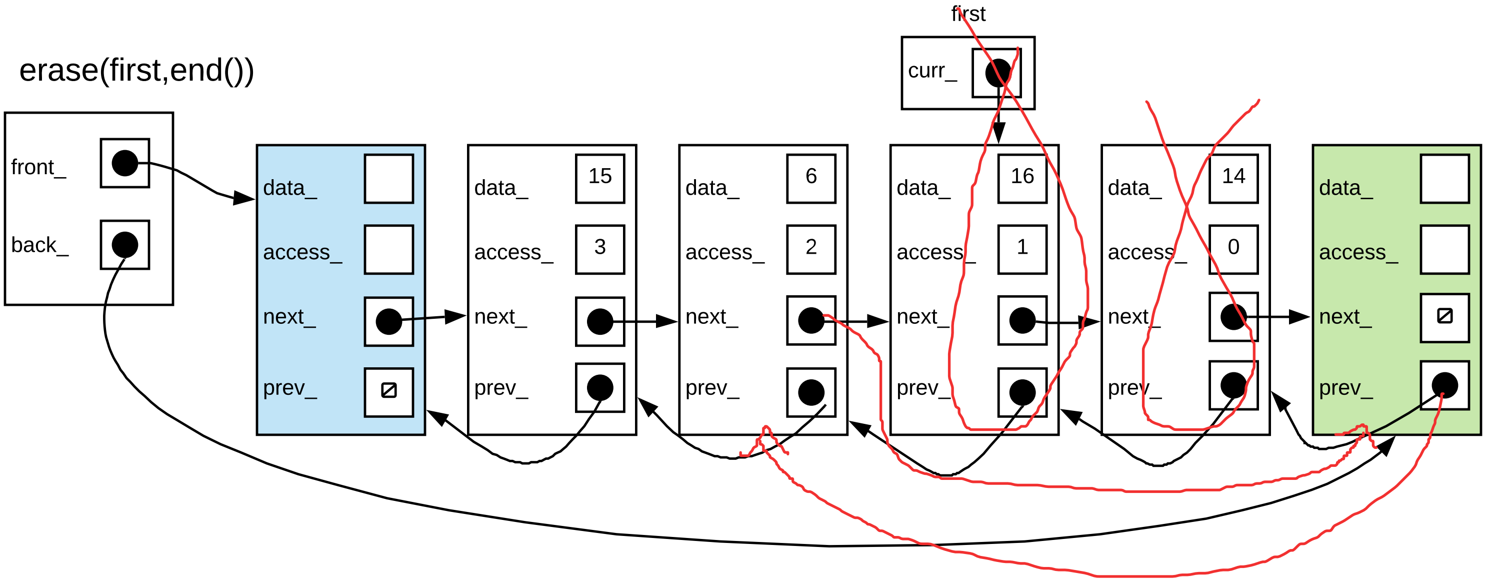
erase(first,last)



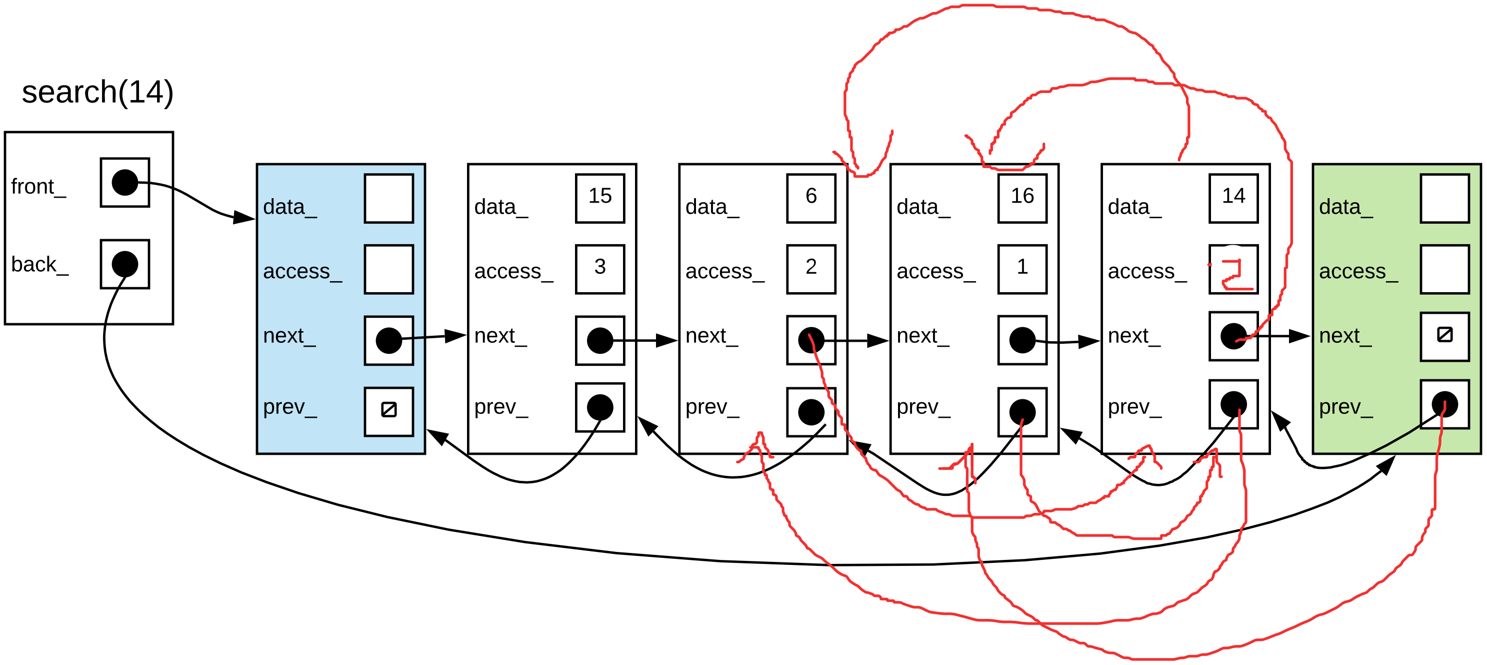
erase(first,last)



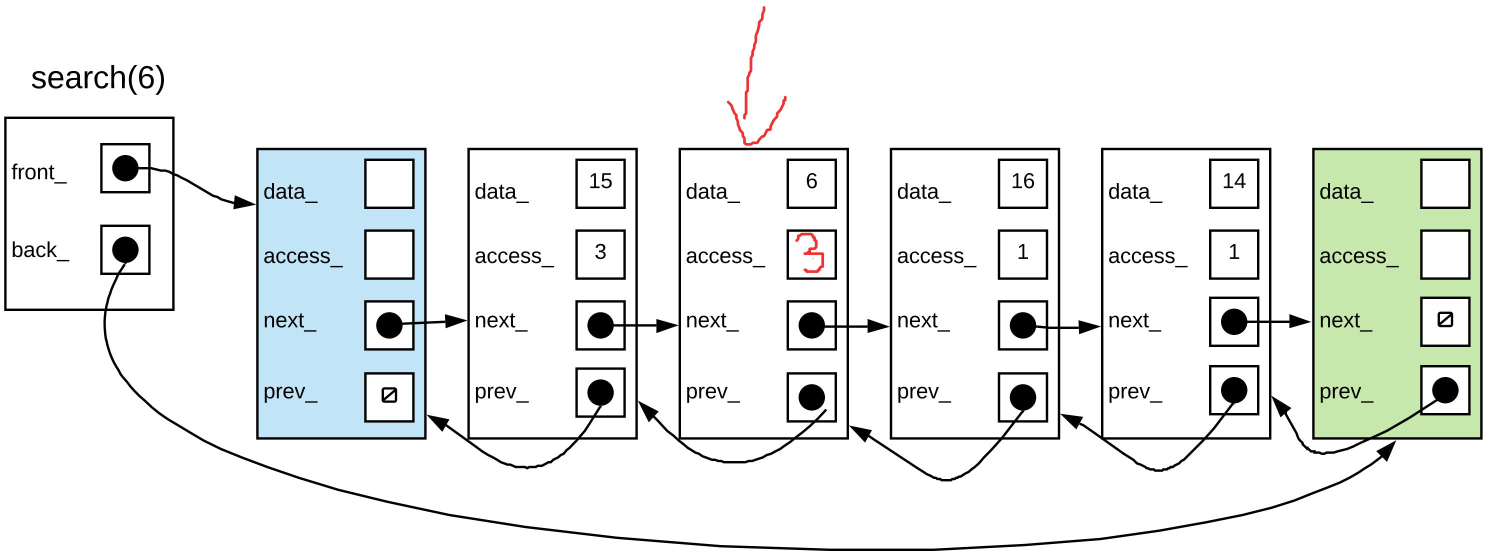
erase(first,end())



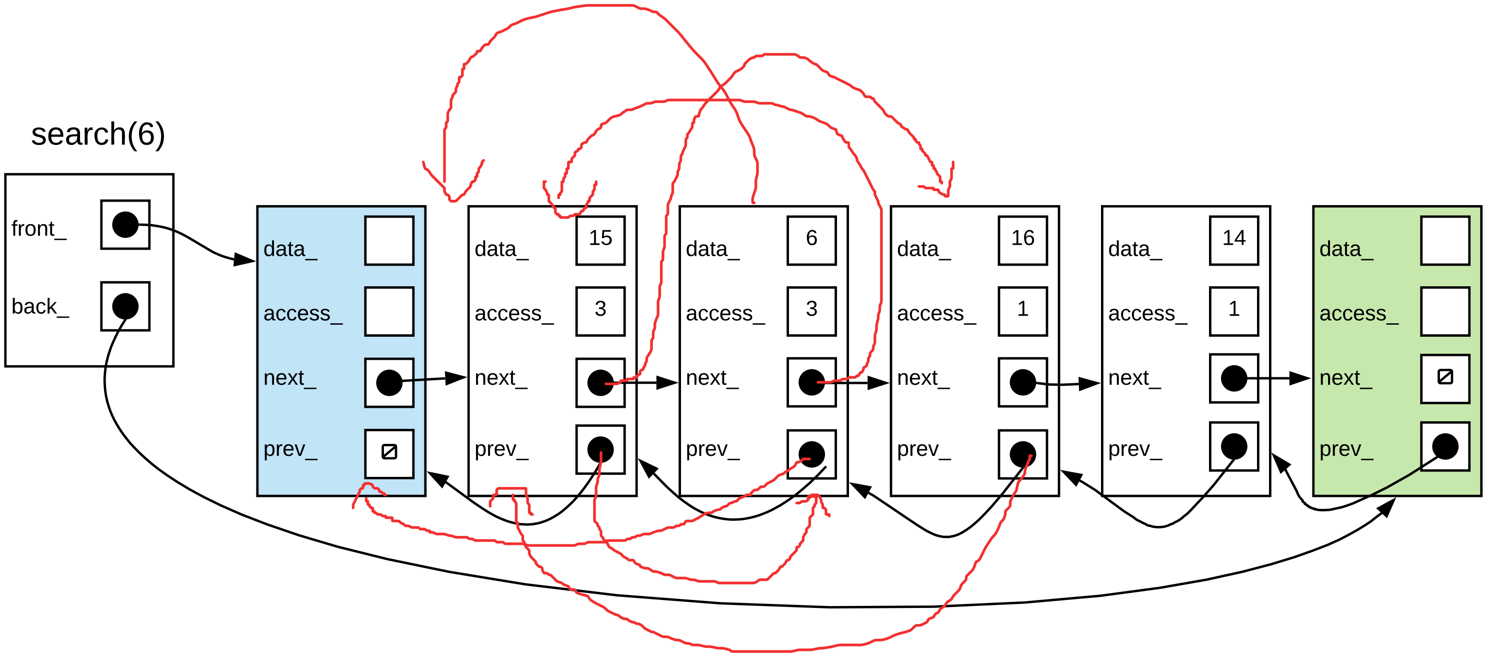
search(14)



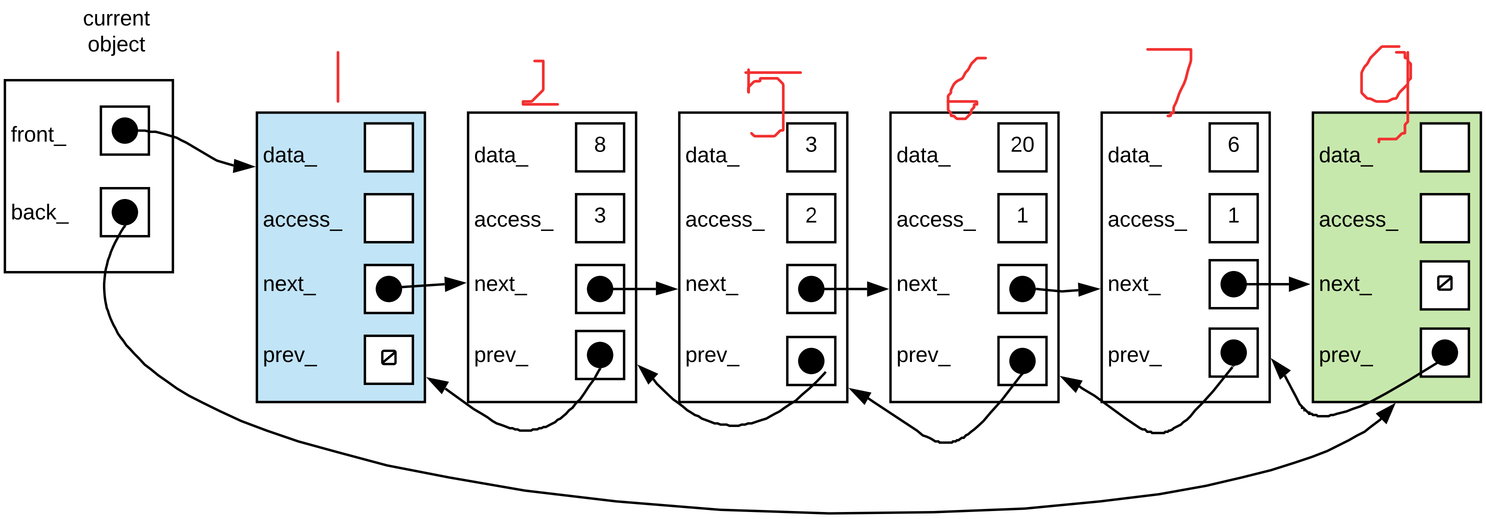
search(6)

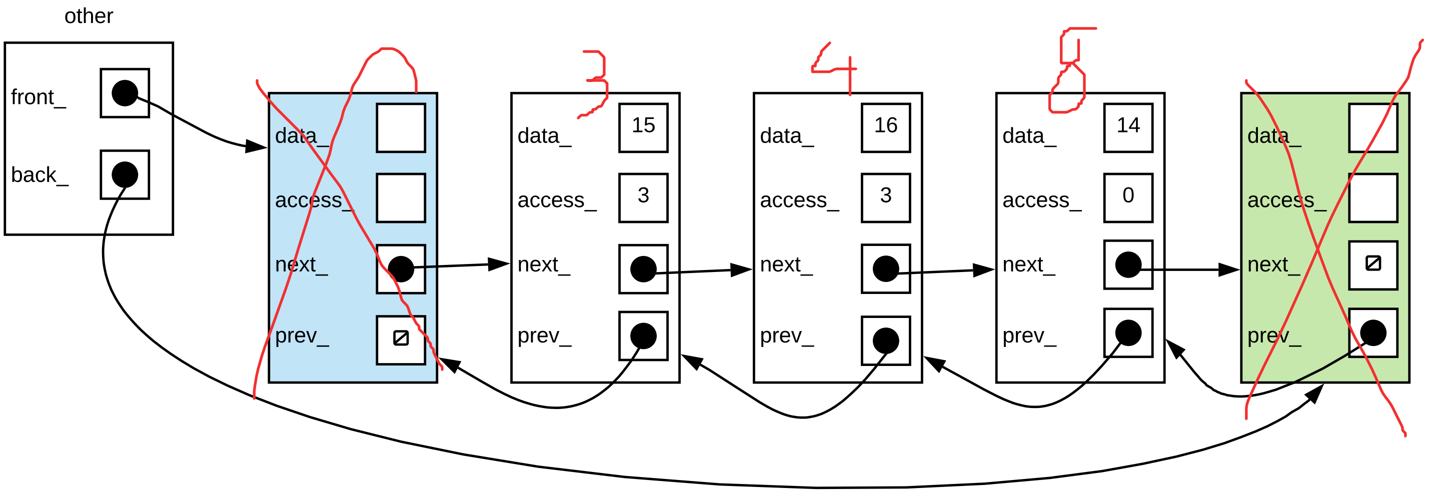


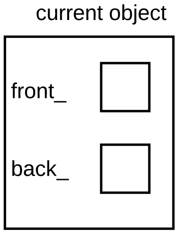
search(6)



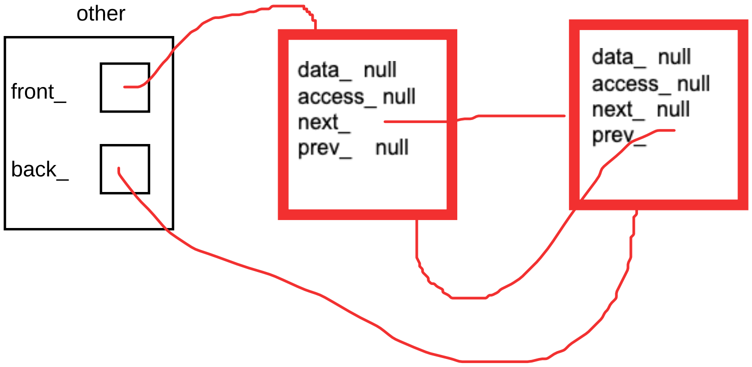
before list.merge(other)

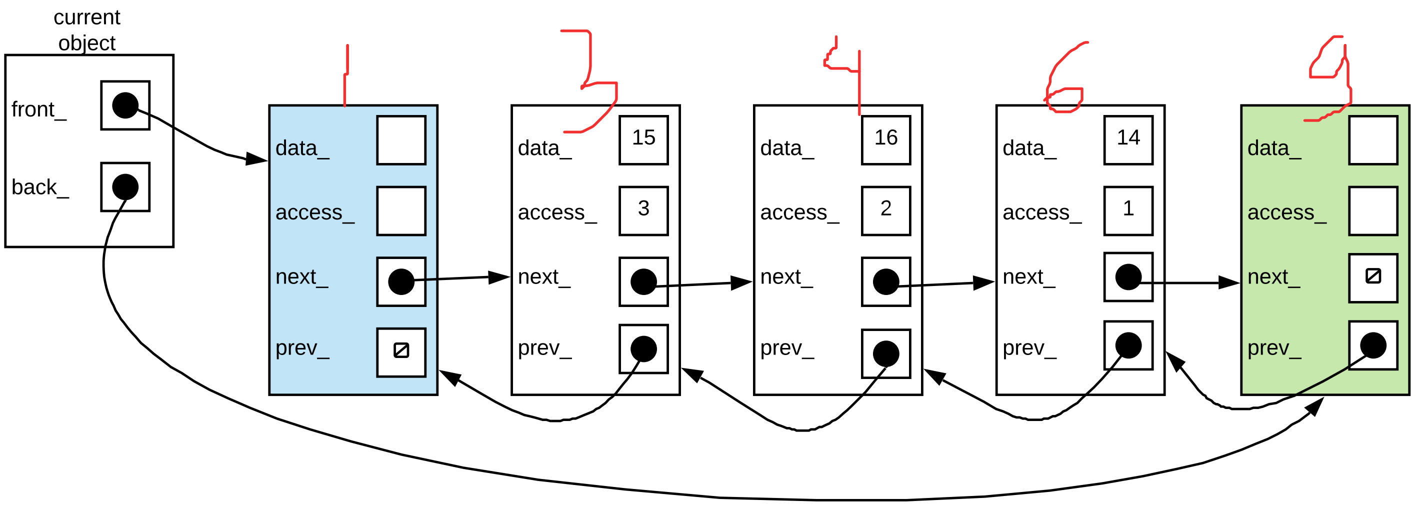


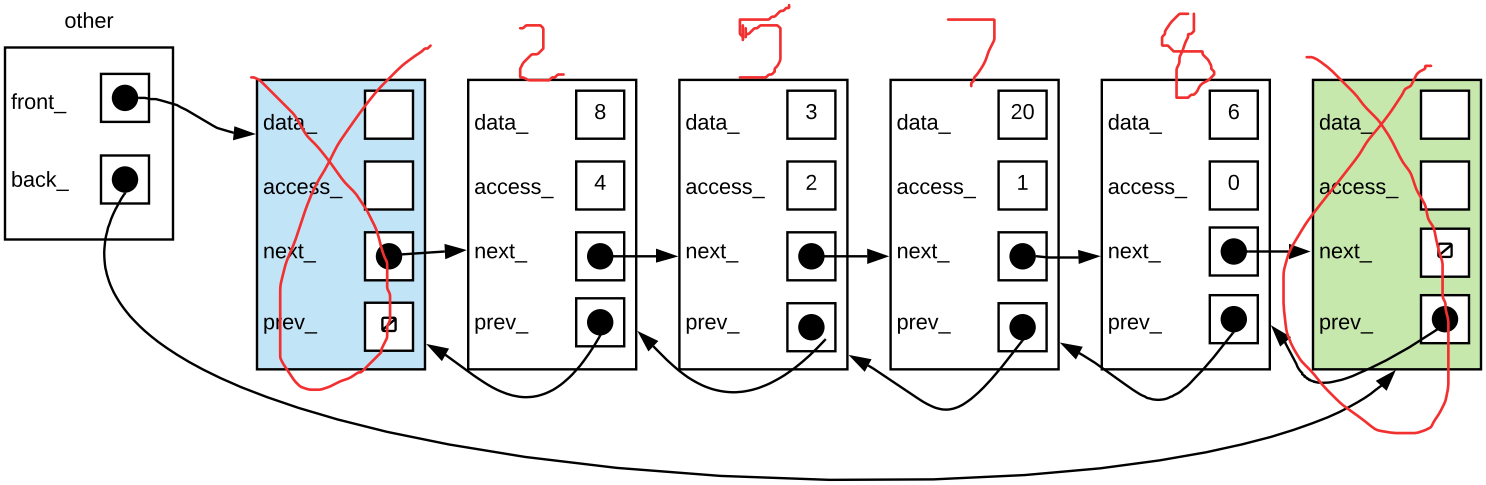


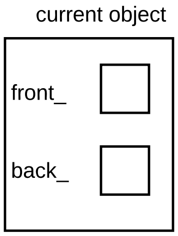


Since List<other> is being merge into current object, the List<other> will be empy and current object will contain extra data from other. Because spacing is limited, I will note the number order above the data

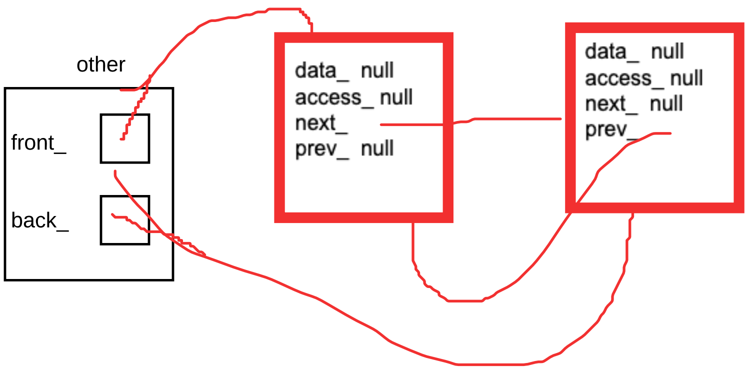




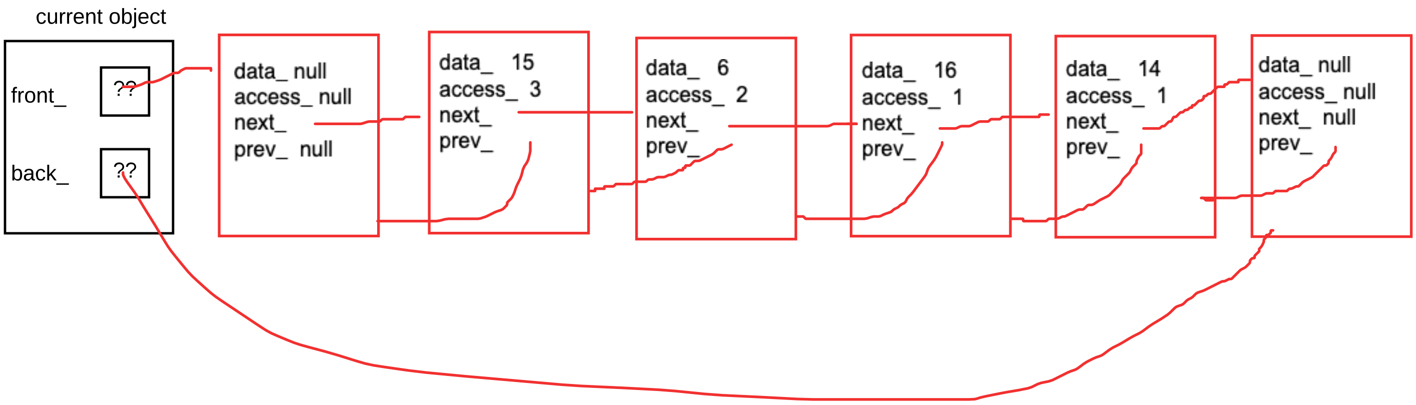


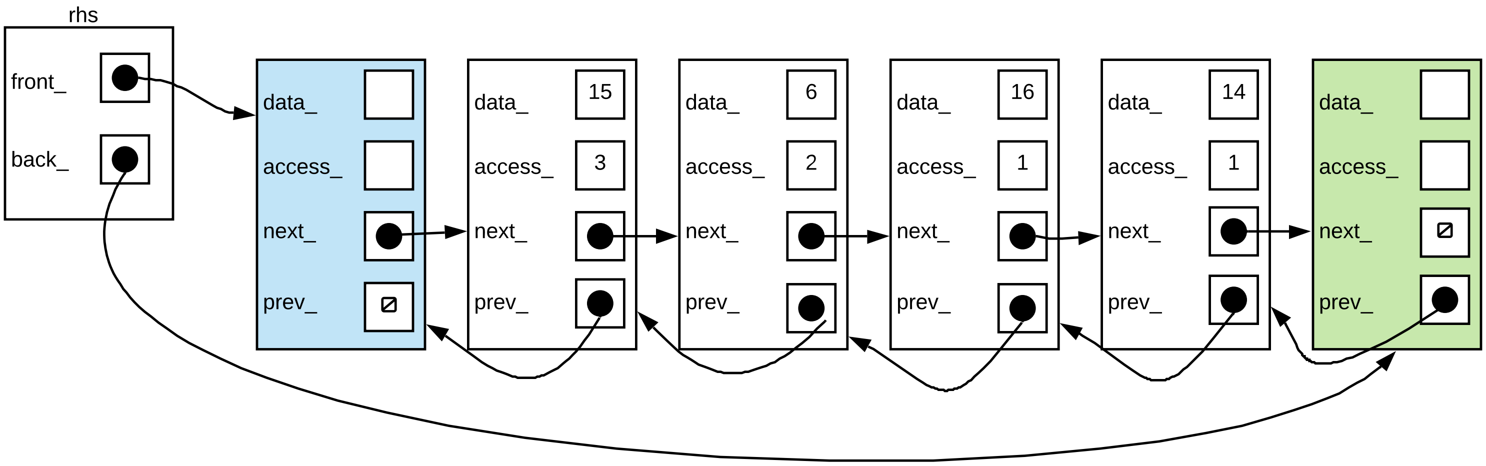


Since List<other> is being merge into current object, the List<other> will be empy and current object will contain extra data from other.
Because spacing is limited, I will note the number order above the data

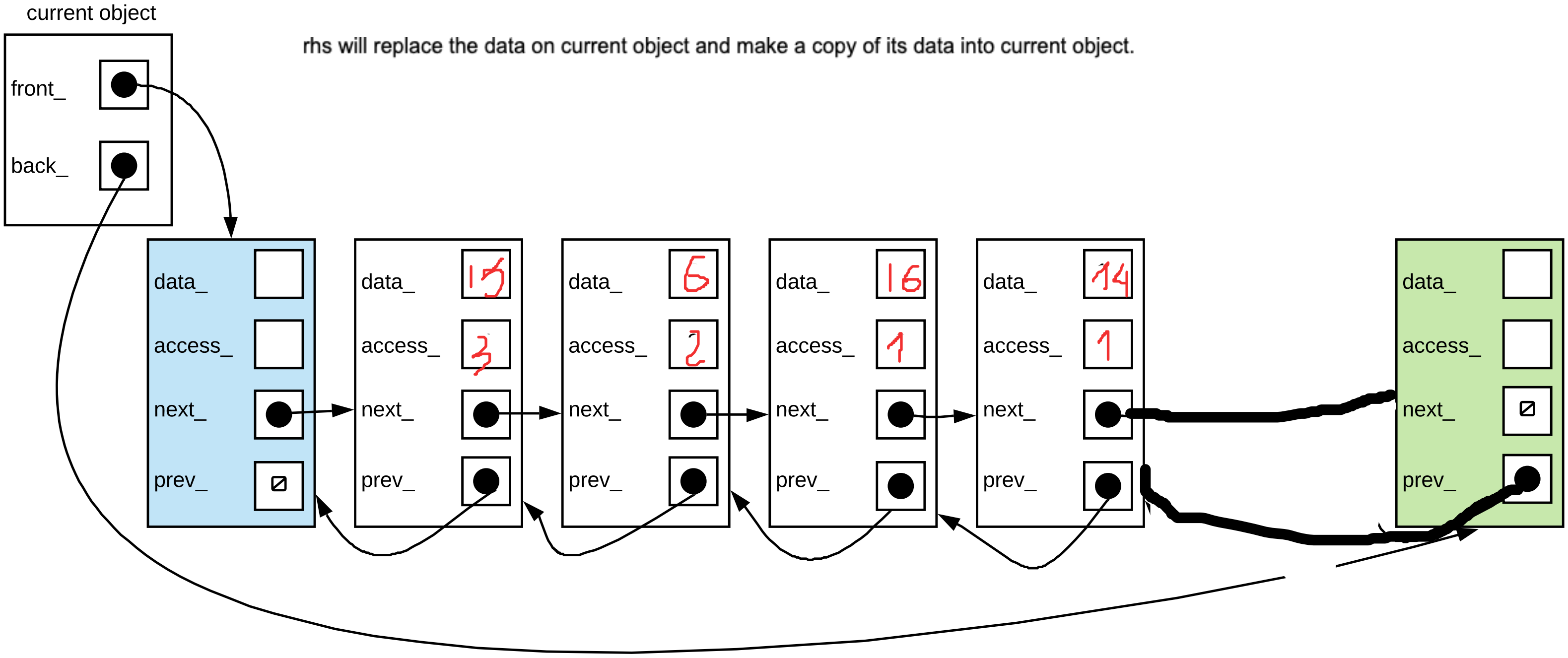


copy constructor -
CacheList(rhs);

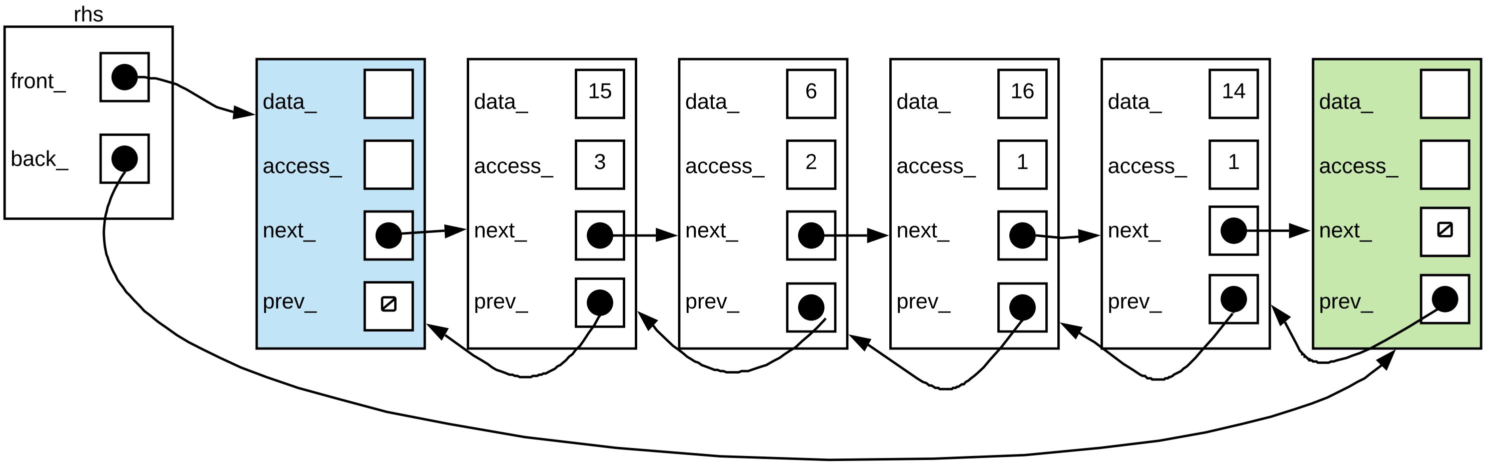




copy assignment

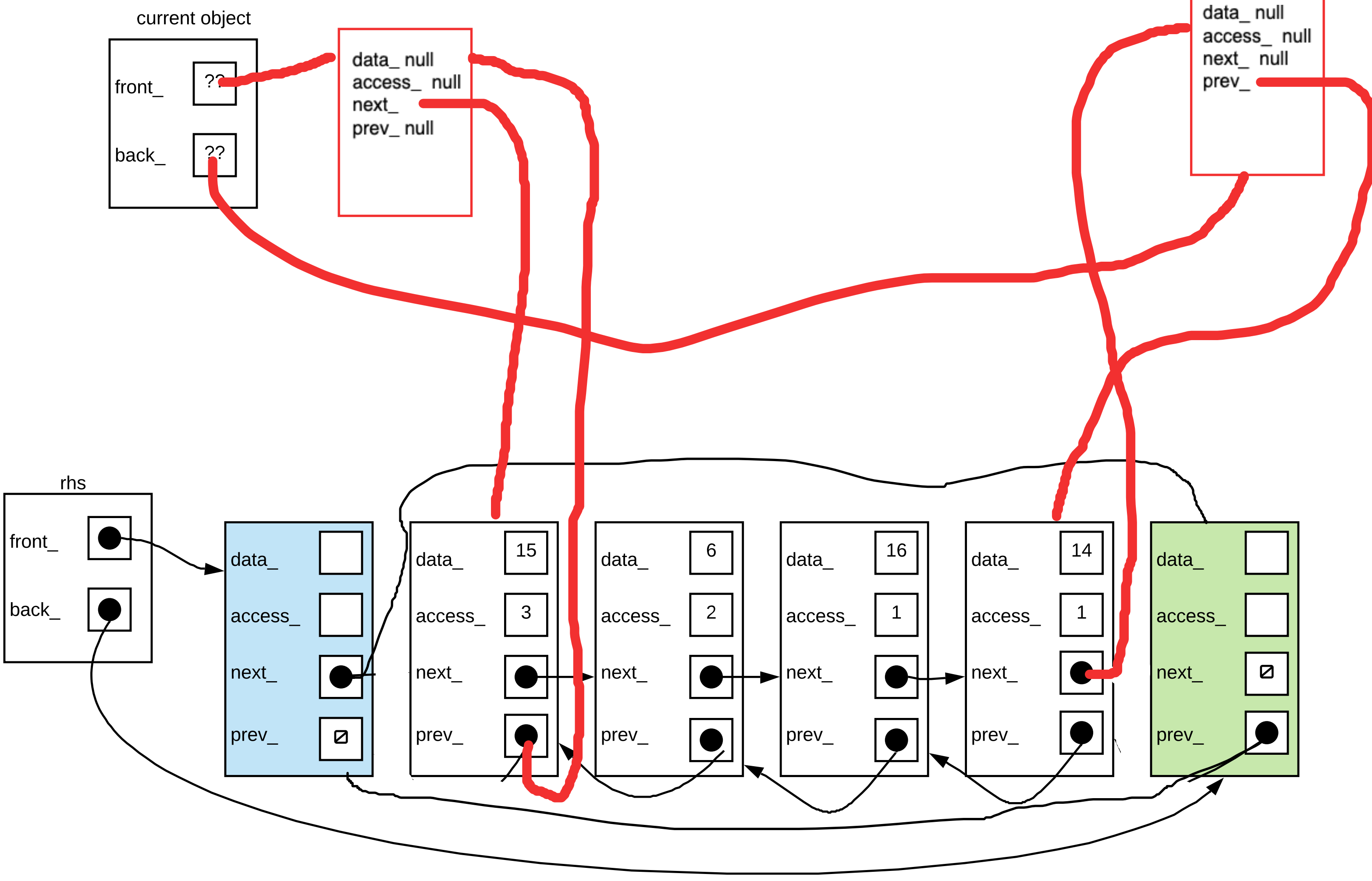


rhs will replace the data on current object and make a copy of its data into current object.



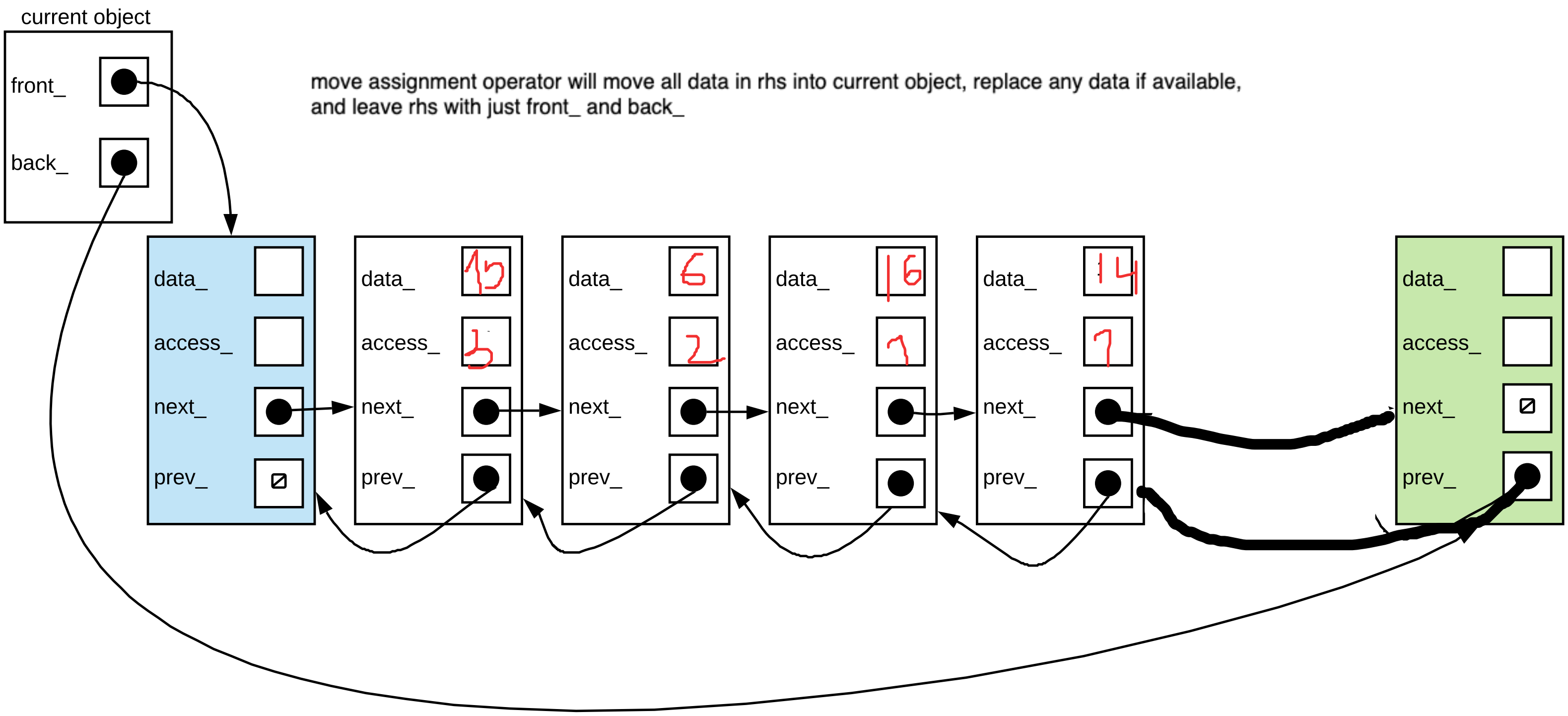
move constructor -
CacheList(rhs);

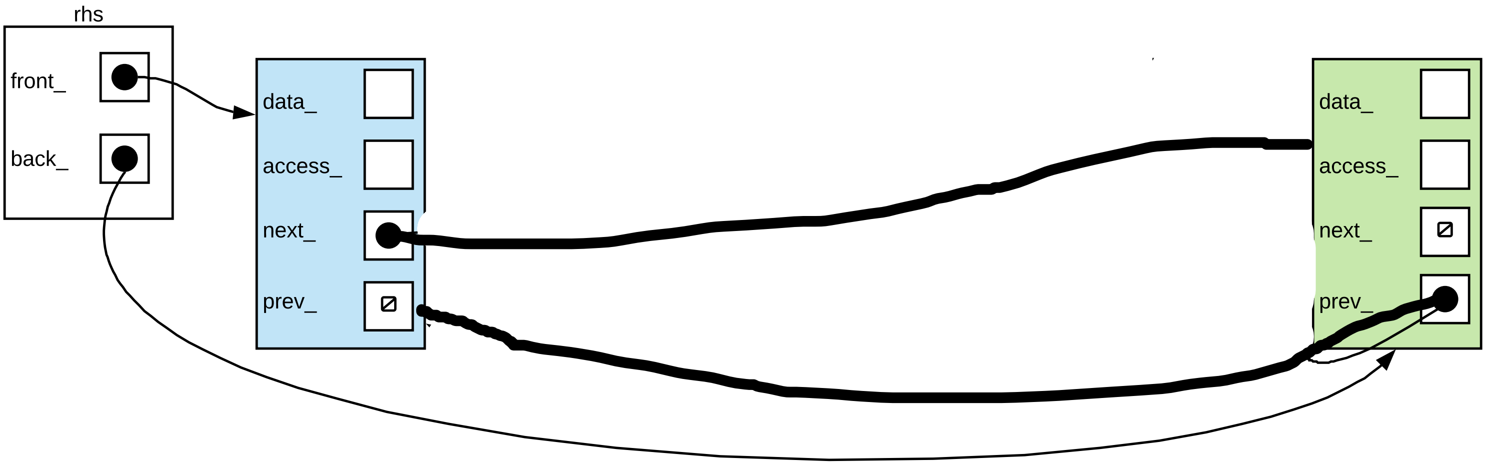
move constructor will move all data from rhs into current object and leave rhs with front_ and back_ node since it is sentinel. Current object will have existing front_ and back_ and even data, will be replaced by data from rhs.



move assignment

move assignment operator will move all data in rhs into current object, replace any data if available, and leave rhs with just front_ and back_


current object
front_
back_
data_
access_
next_
prev_
⊘
data_
15
access_
5
next_
prev_
data_
6
access_
2
next_
prev_
data_
16
access_
1
next_
prev_
data_
14
access_
7
next_
prev_
data_
access_
next_
⊘
prev_


rhs
front_
back_
data_
access_
next_
prev_
⊘
data_
access_
next_
⊘
prev_