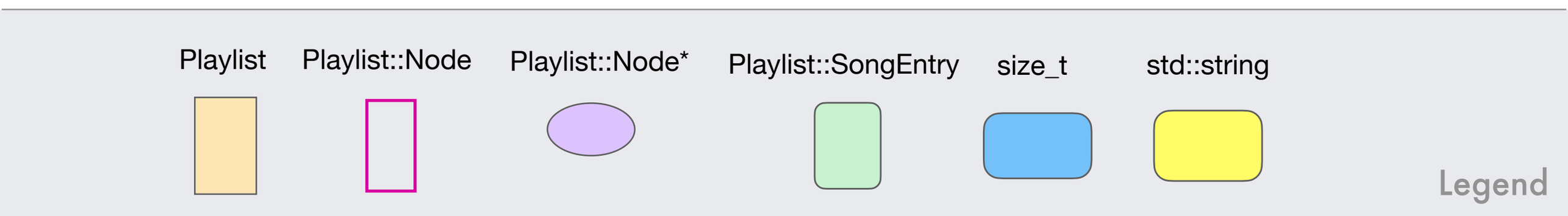
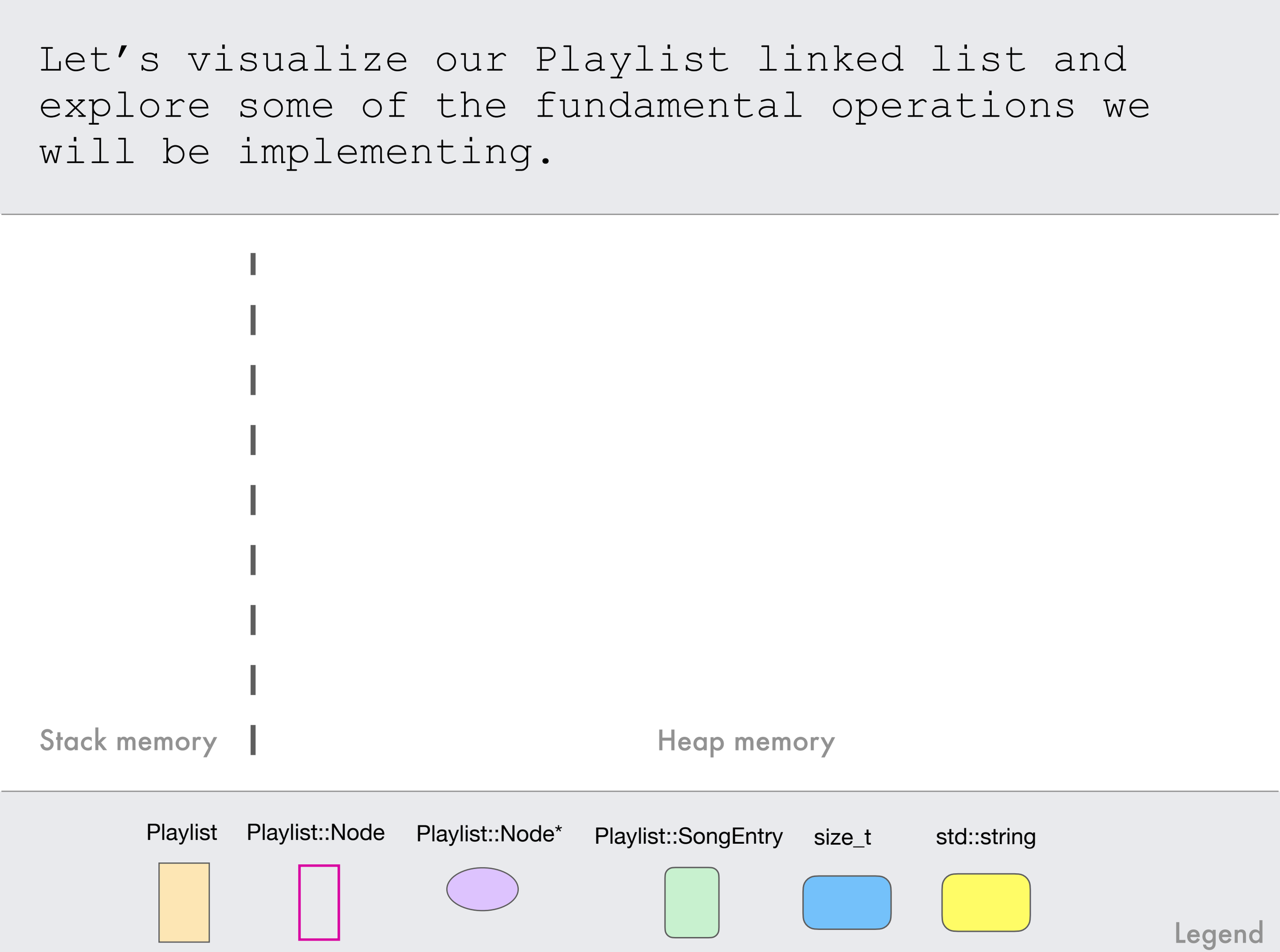
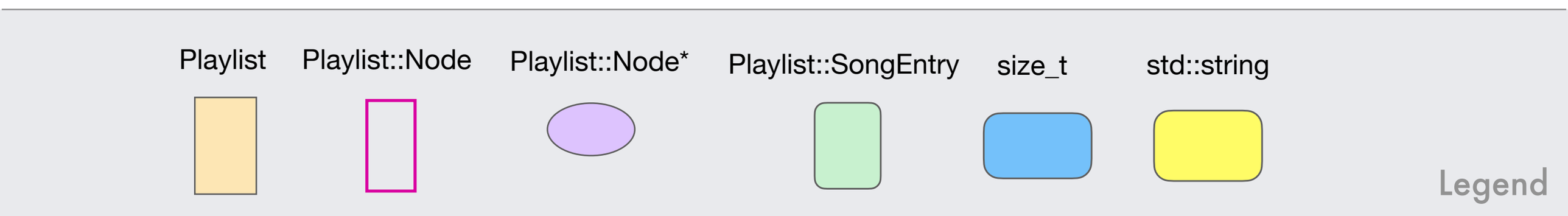
CS2B Project 02: Playlist
* Click to advance, or use arrow keys
* Hover far left for thumbnail sidebar



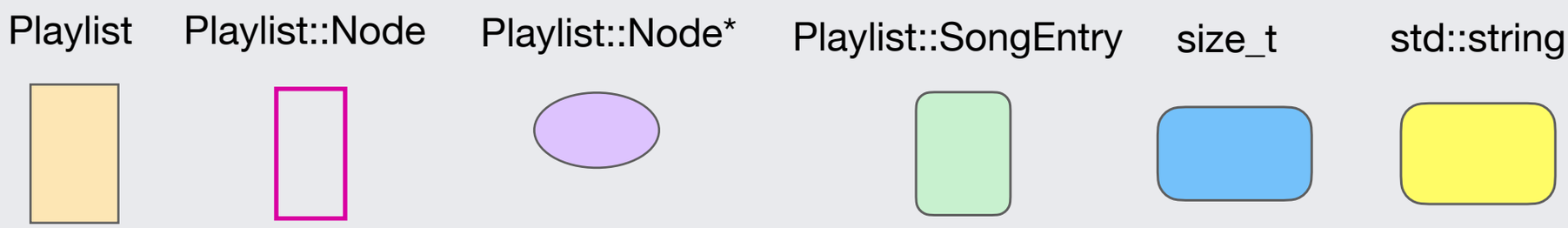
Let’s visualize our Playlist linked list and explore some of the fundamental operations we will be implementing.



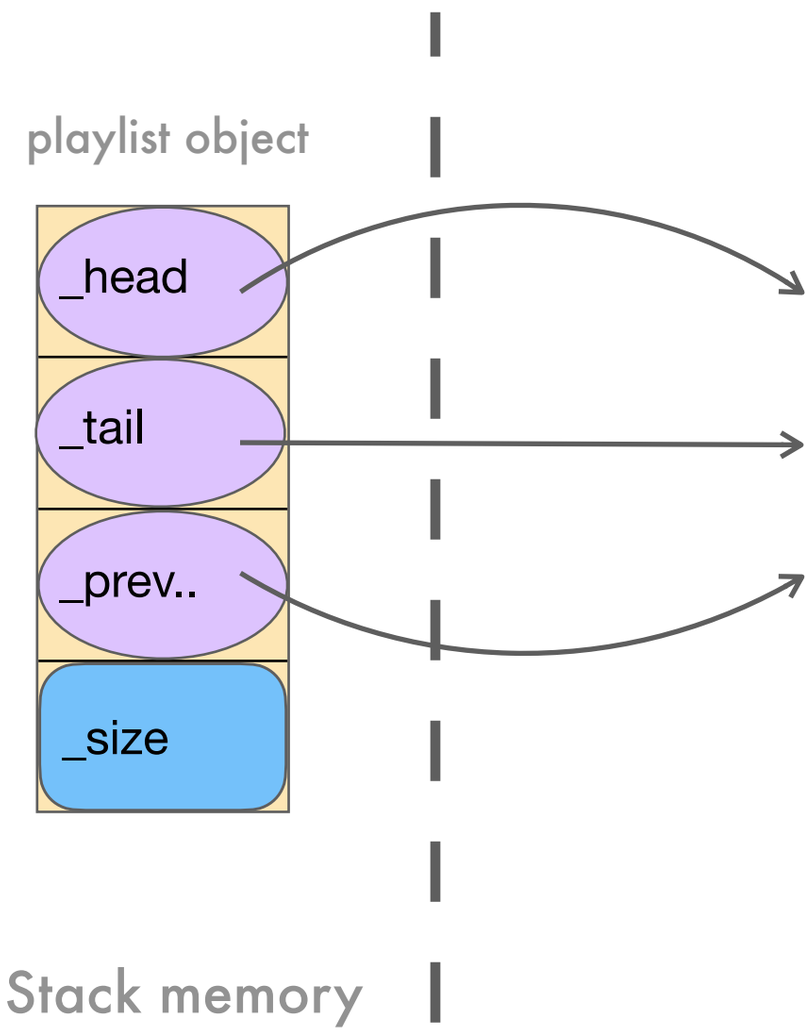
We'll consider a local playlist object created by the user that lives in stack memory.

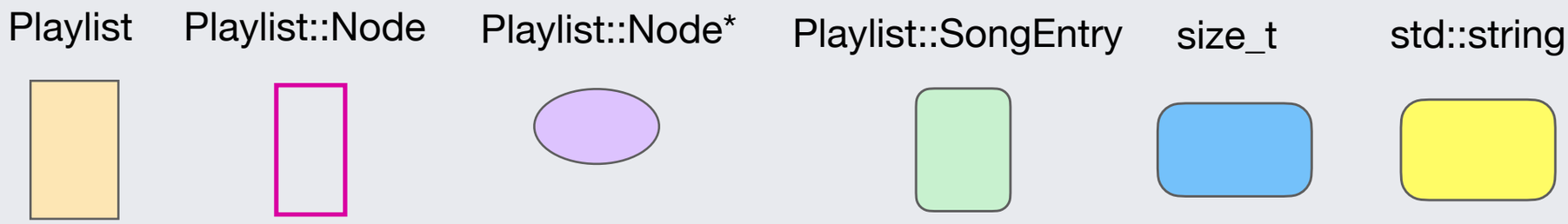


We'll consider a local playlist object created by the user that lives in stack memory.

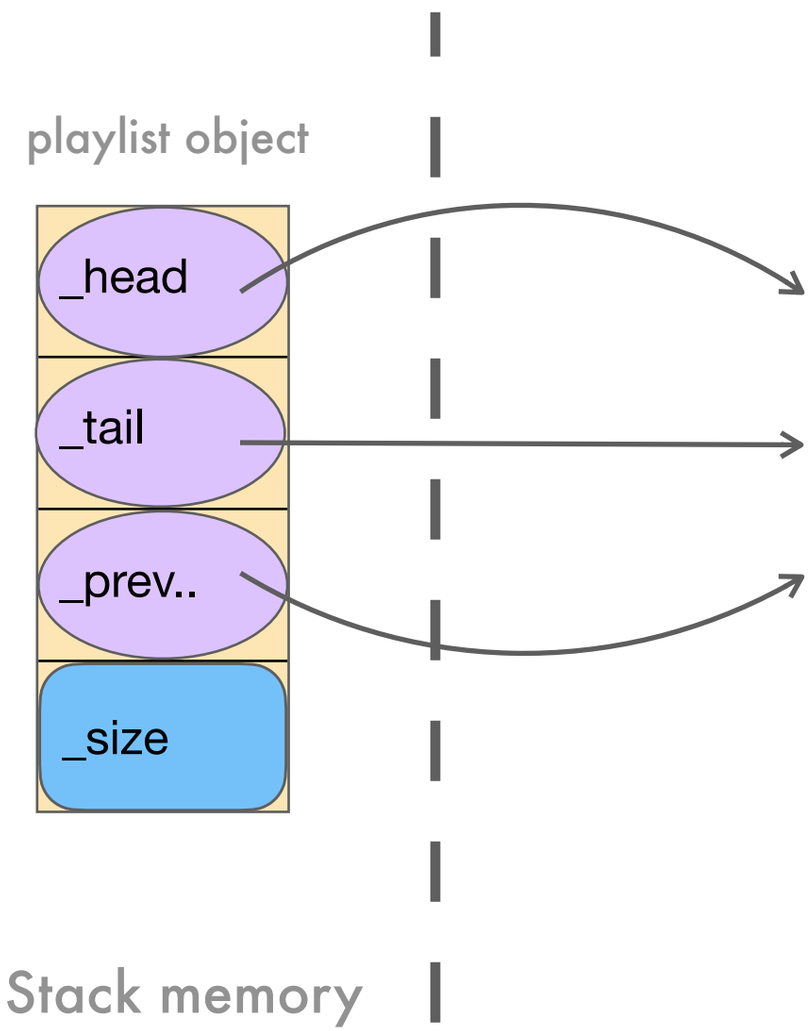


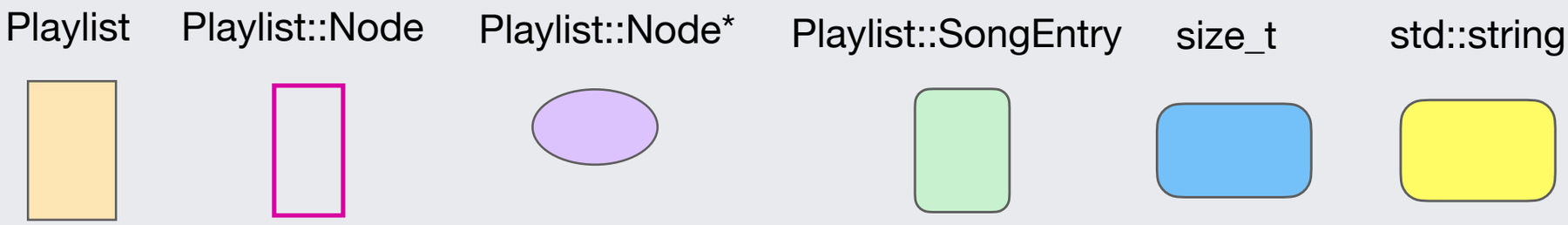
It will have node pointer (Playlist::Node*) data members that point into heap memory. The Playlist class will dynamically allocate (and deallocate) memory for Nodes in heap memory.



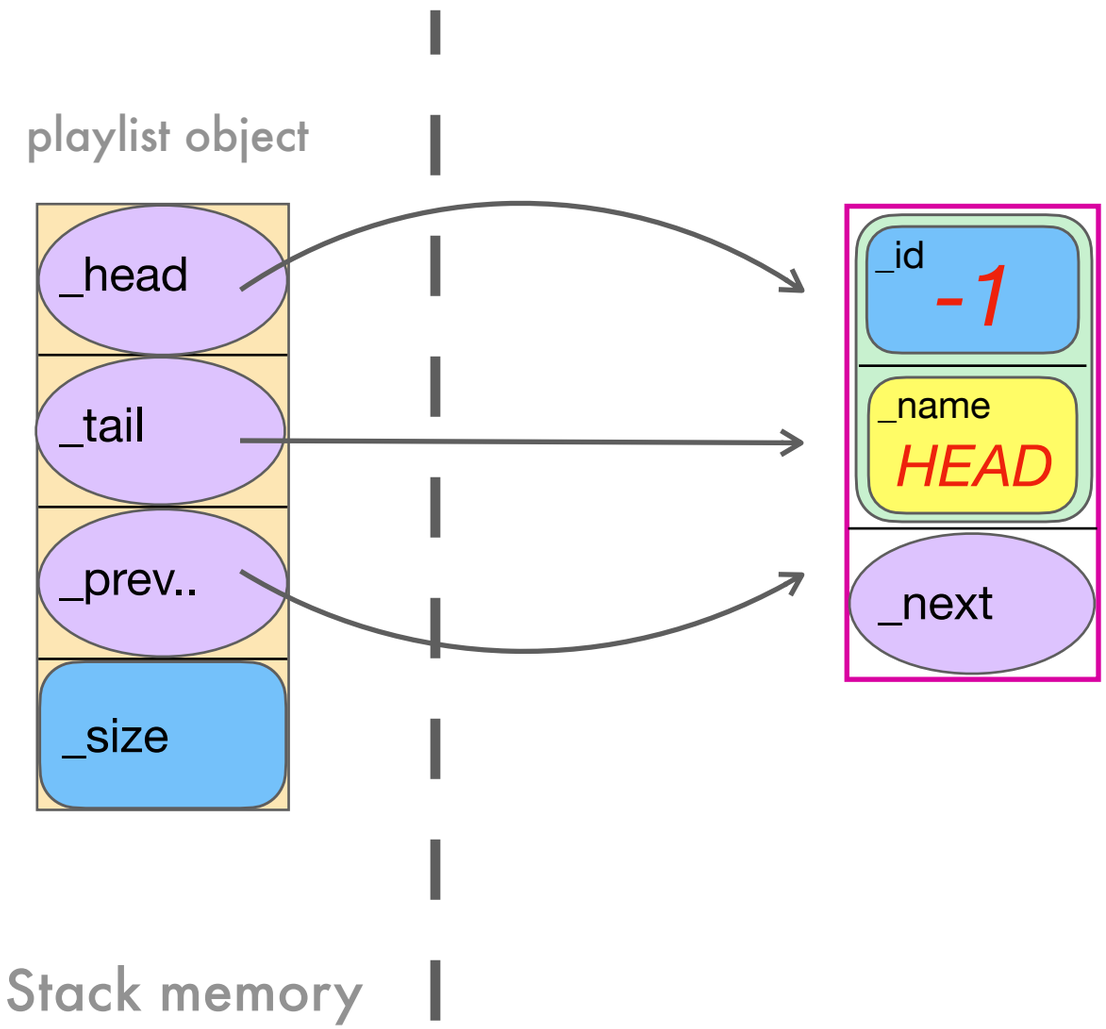


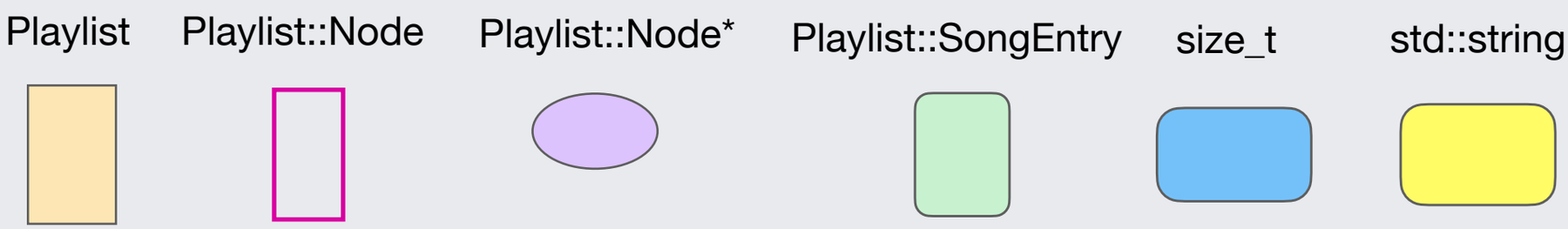
These Node*'s should be initialized to point to the one and same head sentinel node with sentinel SongEntry member values of -1 and "HEAD".



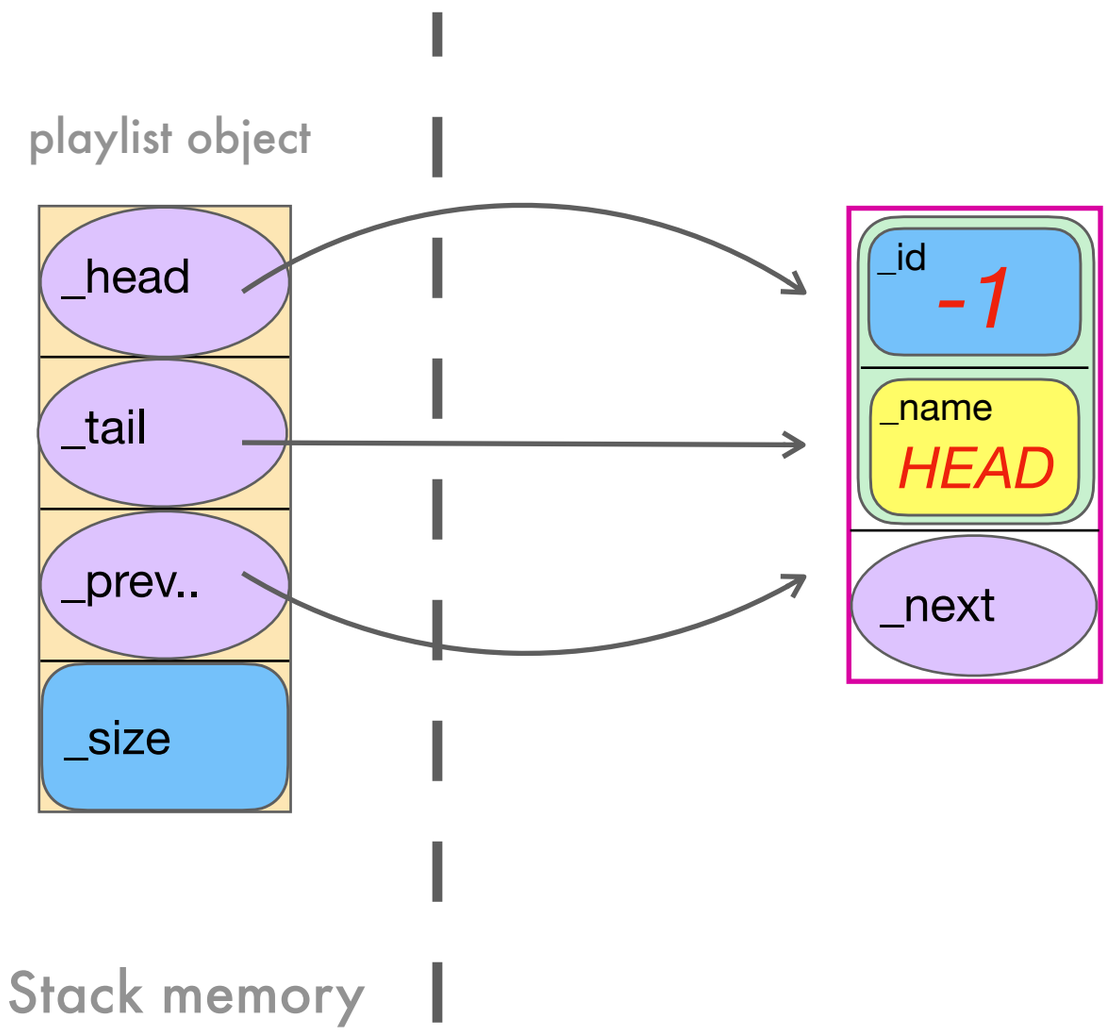


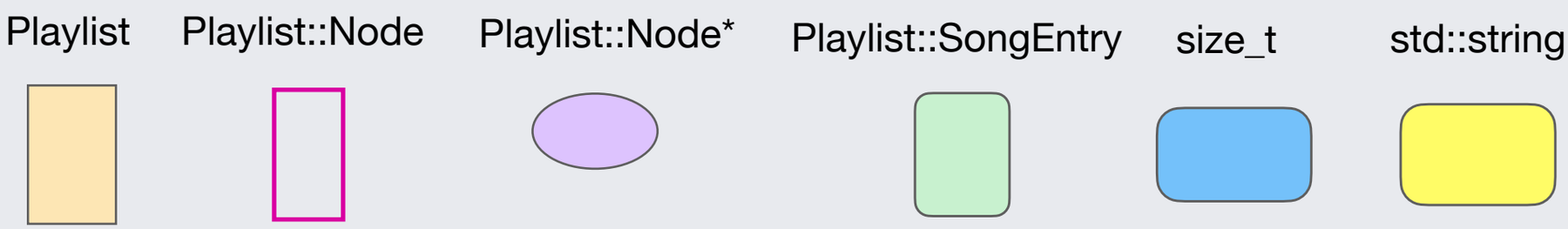
These Node*'s should be initialized to point to the one and same head sentinel node with sentinel SongEntry member values of -1 and "HEAD".



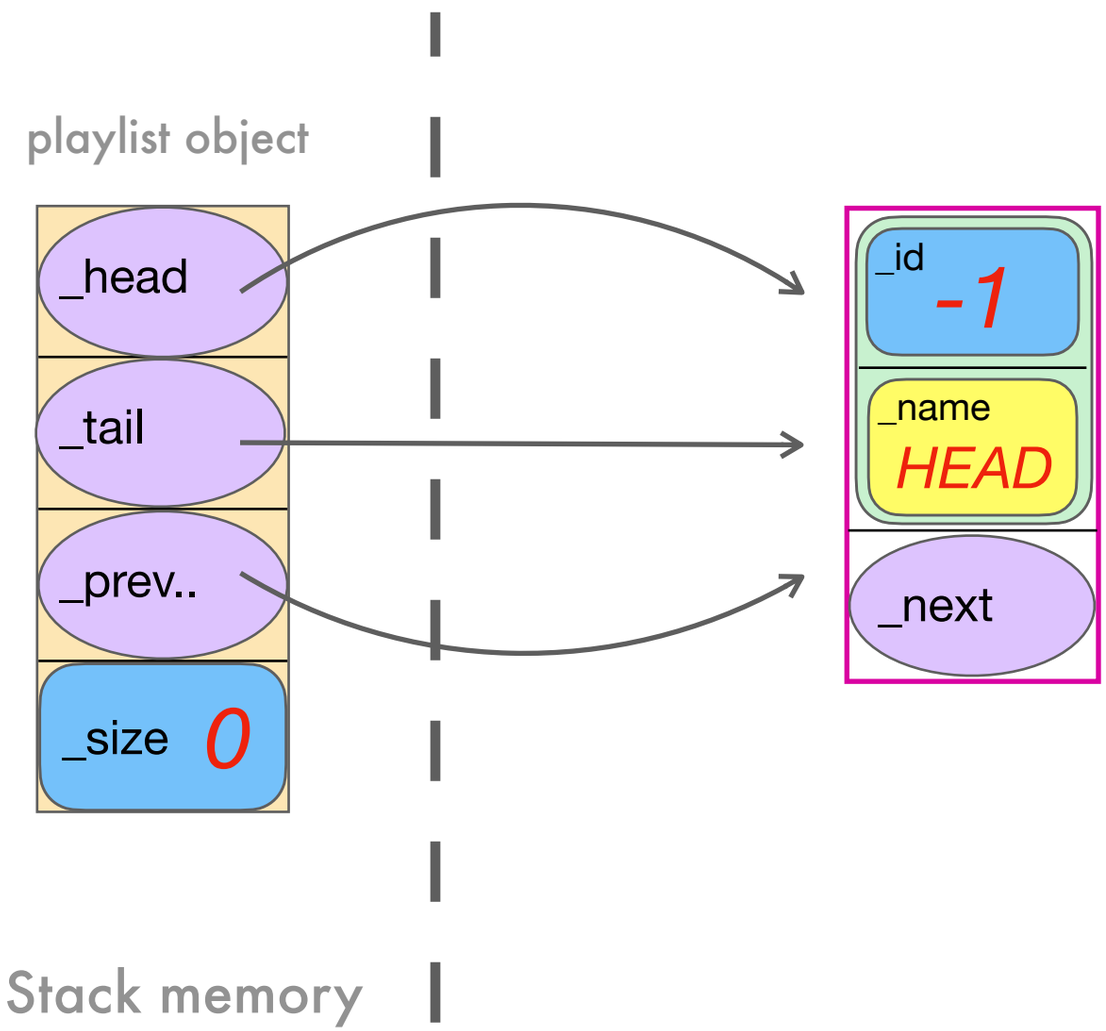


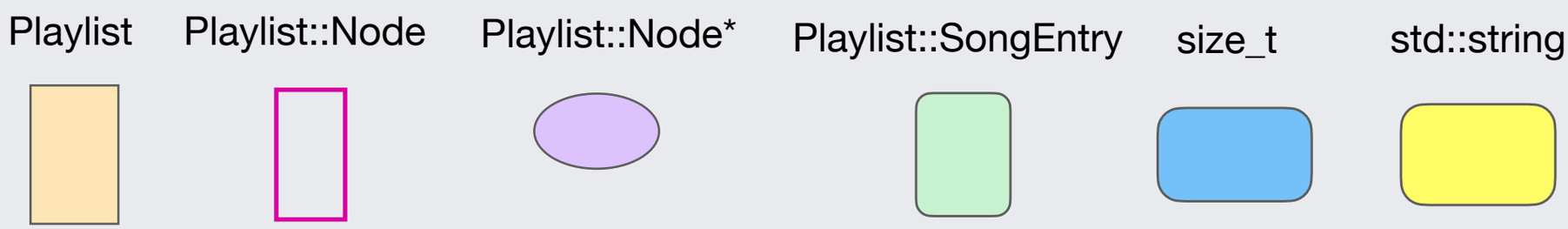
The head sentinel node is an implementation detail and does not represent any user data. Accordingly, _size should initialize to 0.



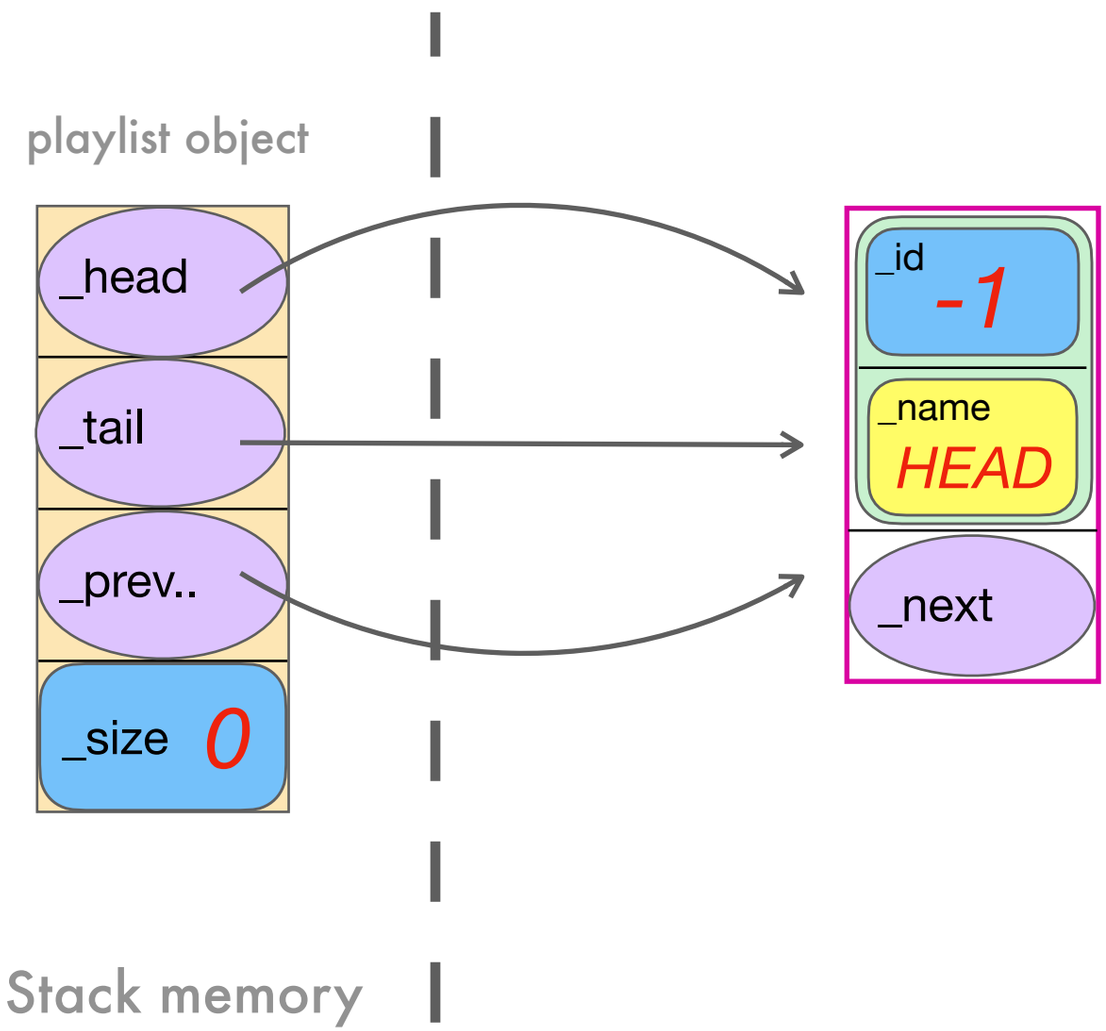


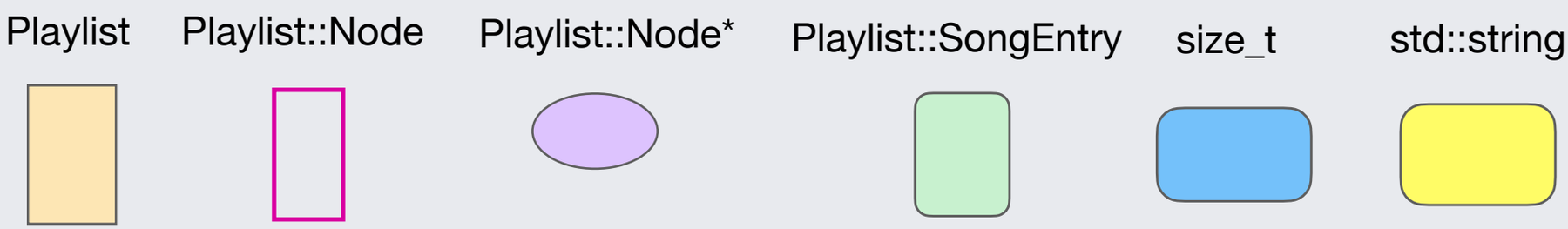
The head sentinel node is an implementation detail and does not represent any user data. Accordingly, _size should initialize to 0.



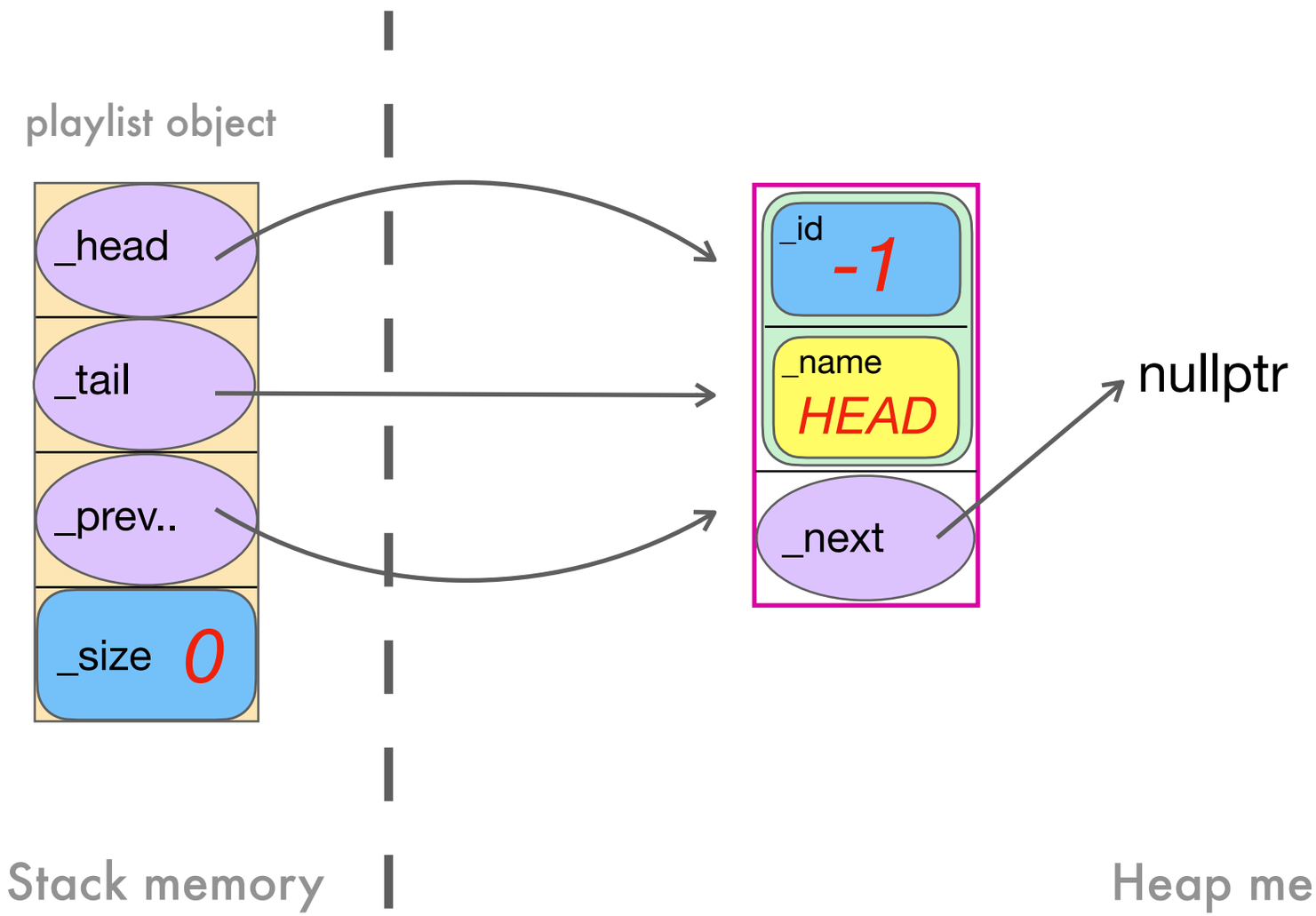


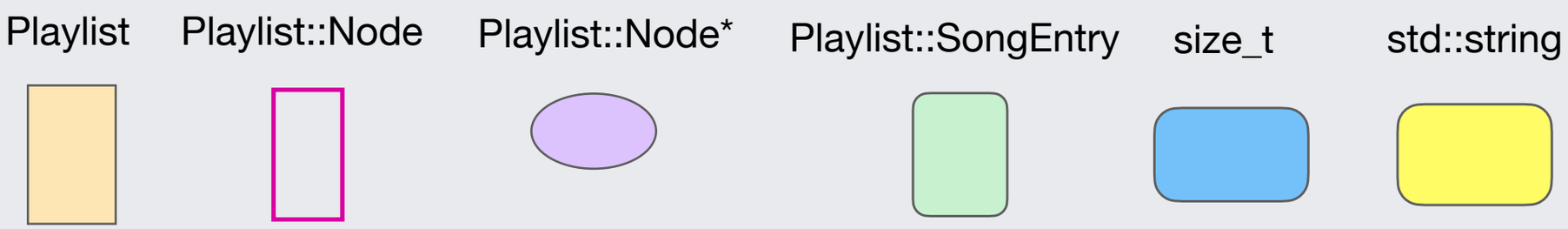
The _next Node* data member should be initialized to nullptr by the Node constructor to avoid pointing at wild (garbage) memory.





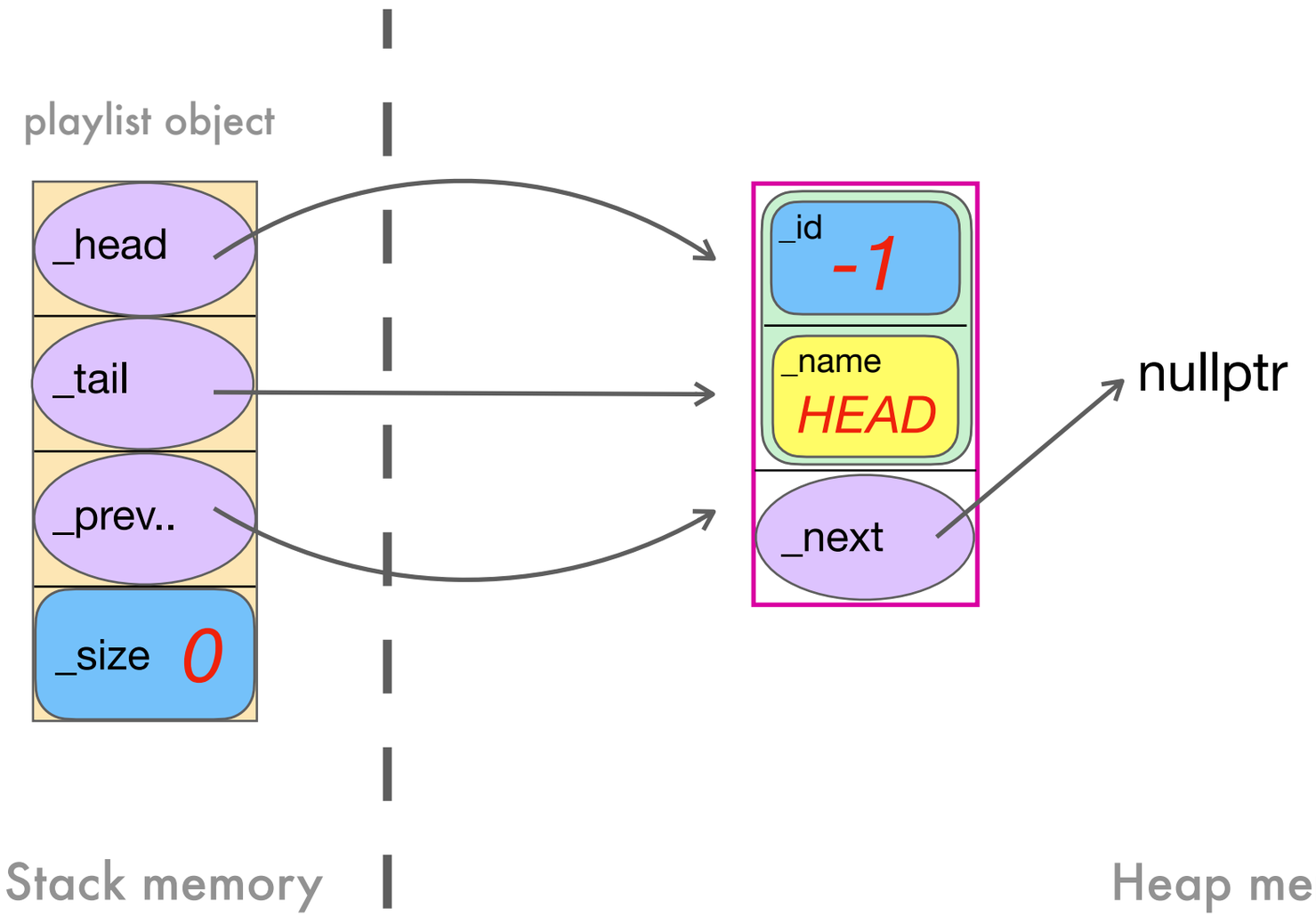
The _next Node* data member should be initialized to nullptr by the Node constructor to avoid pointing at wild (garbage) memory.

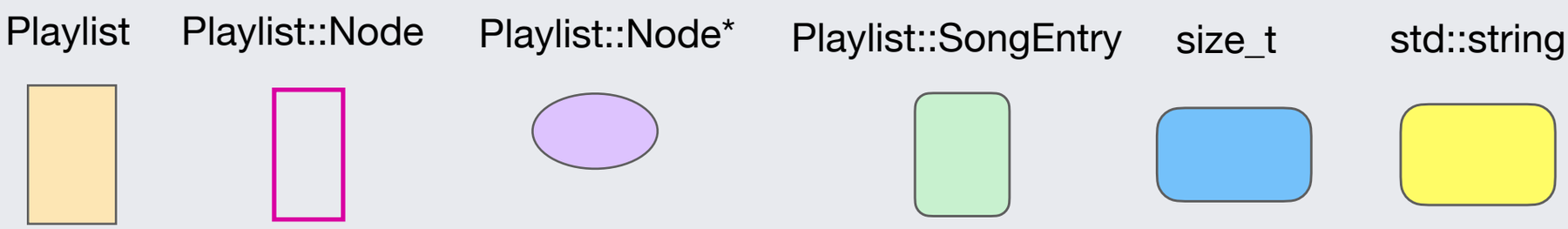




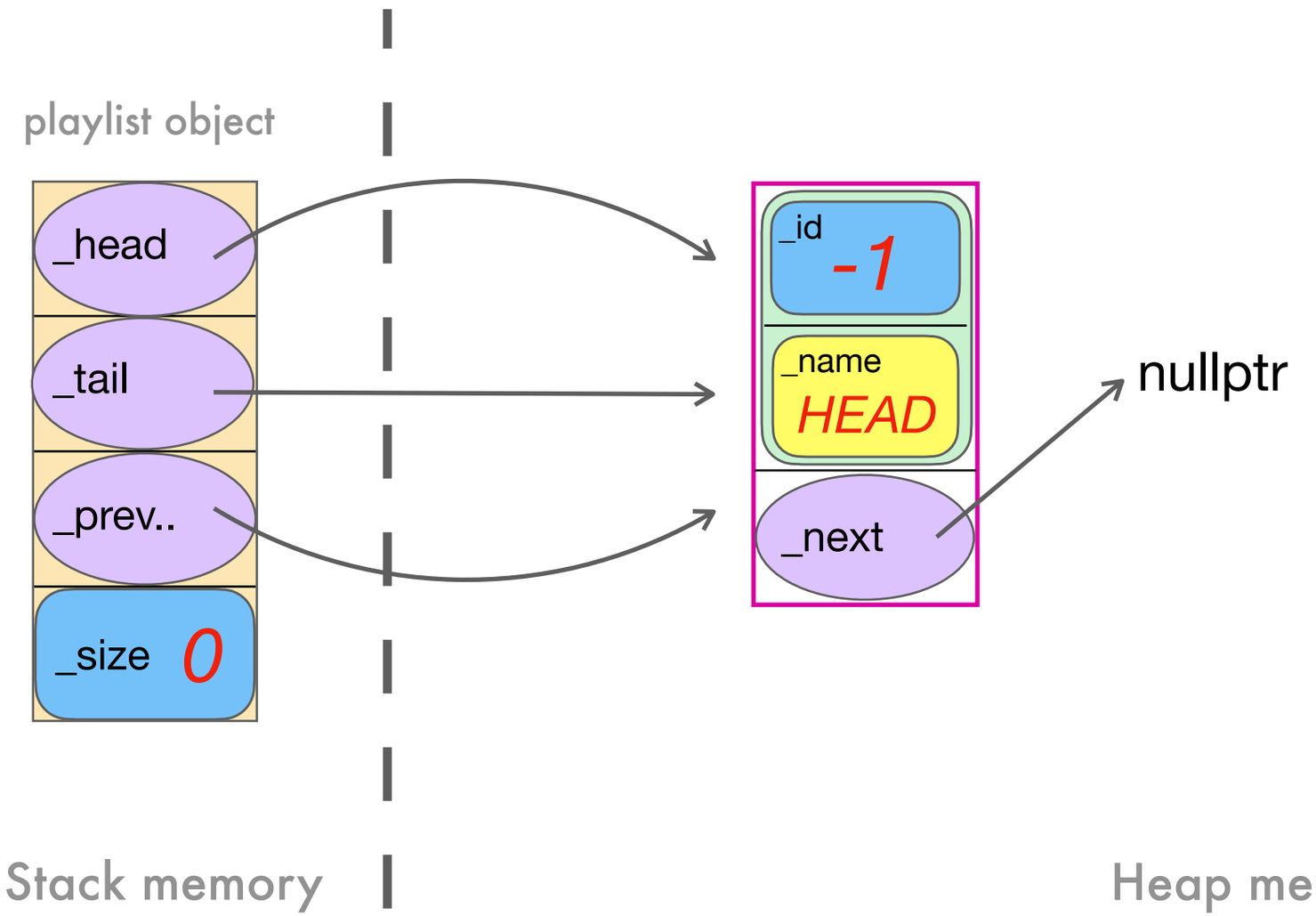
Now, let’s insert some data:

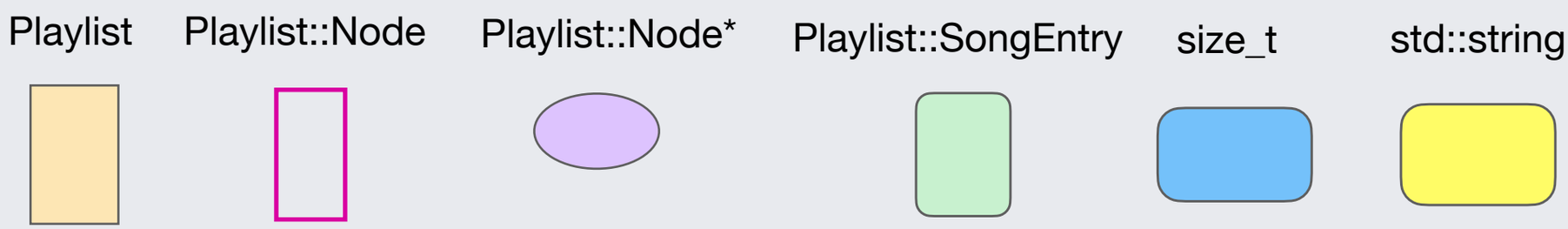
```
playlist.insert_at_cursor(Playlist::SongEntry(10, “A”));
```



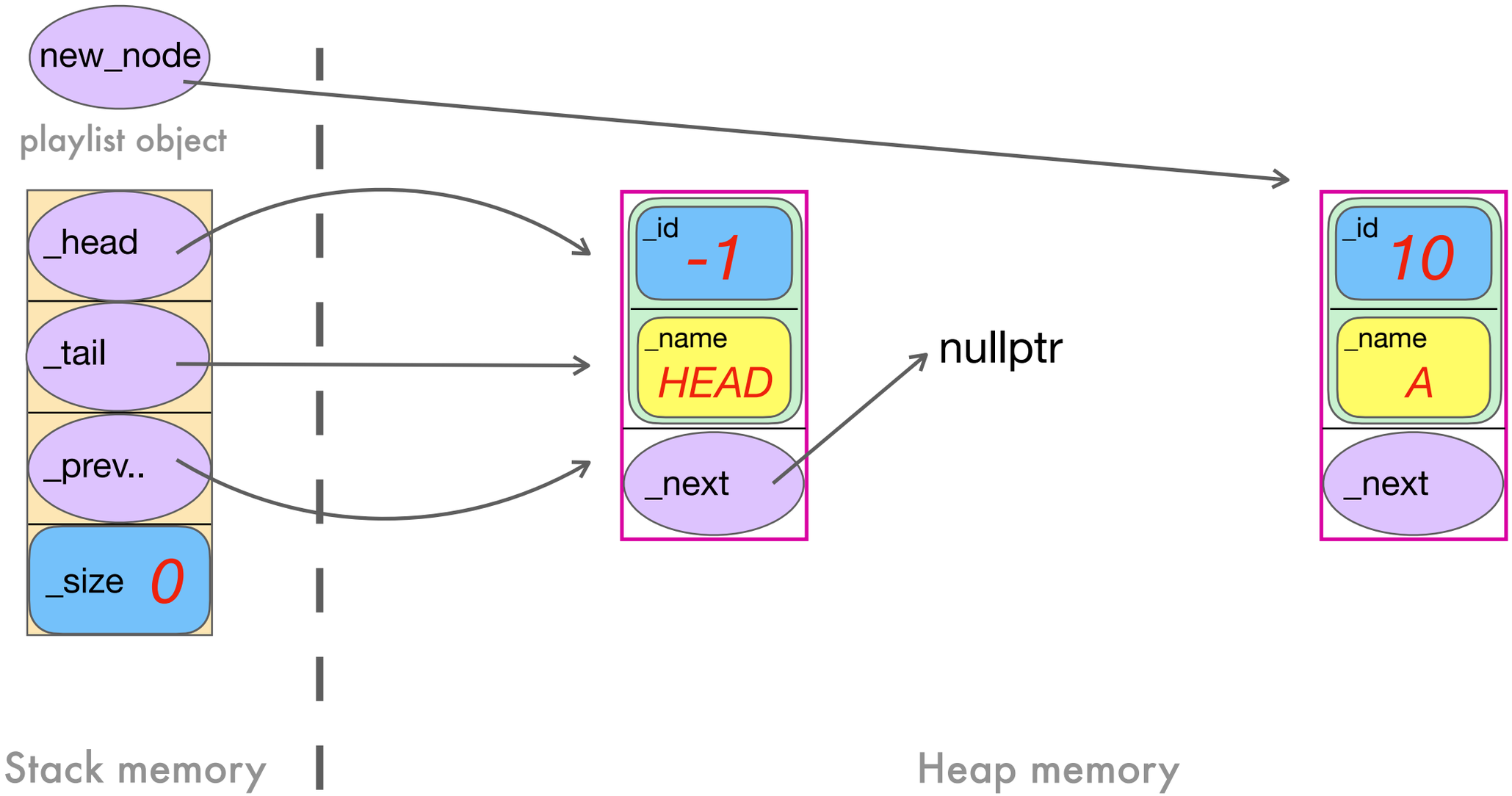


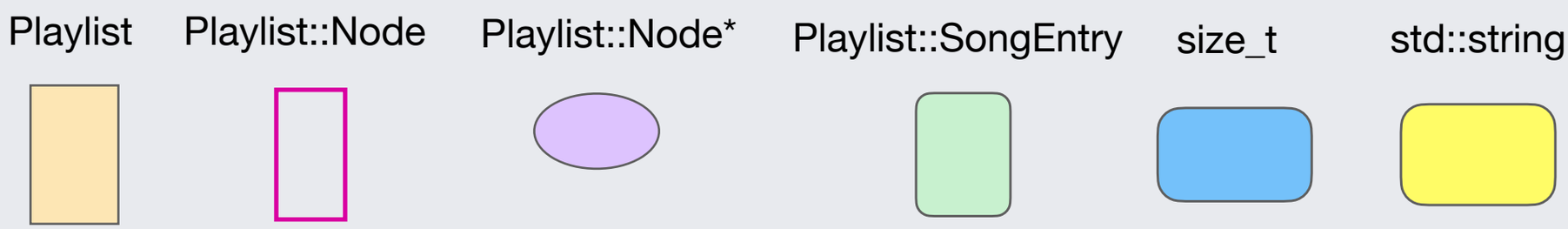
`insert_at_cursor()` will "new up" (allocate memory for) a new node which contains the user's SongEntry..



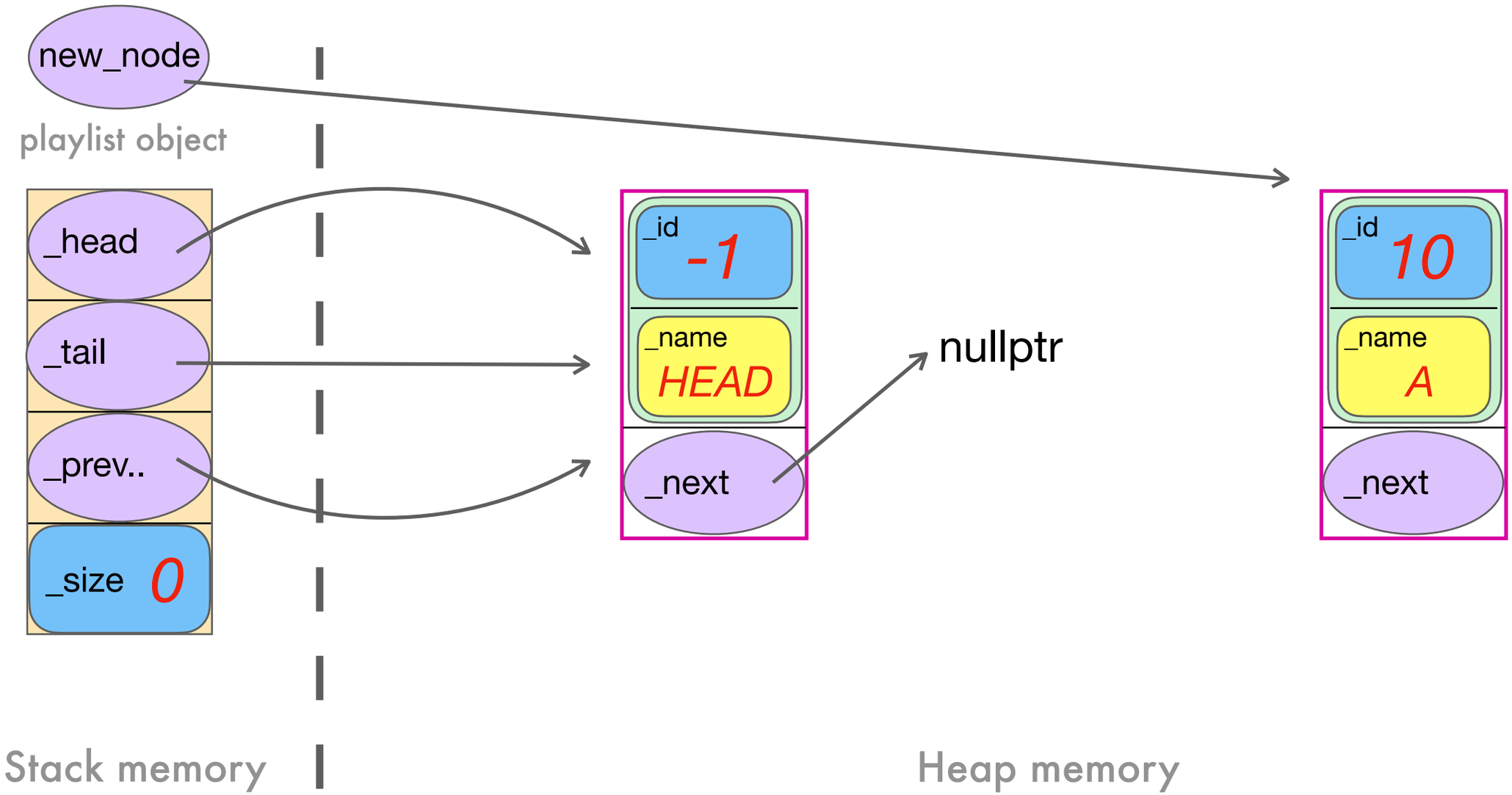


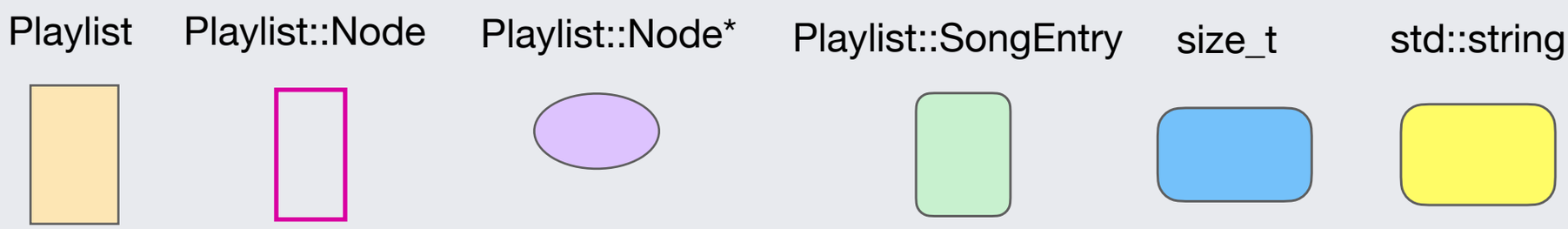
insert_at_cursor() will “new up” (allocate memory for) a new node which contains the user’s SongEntry..



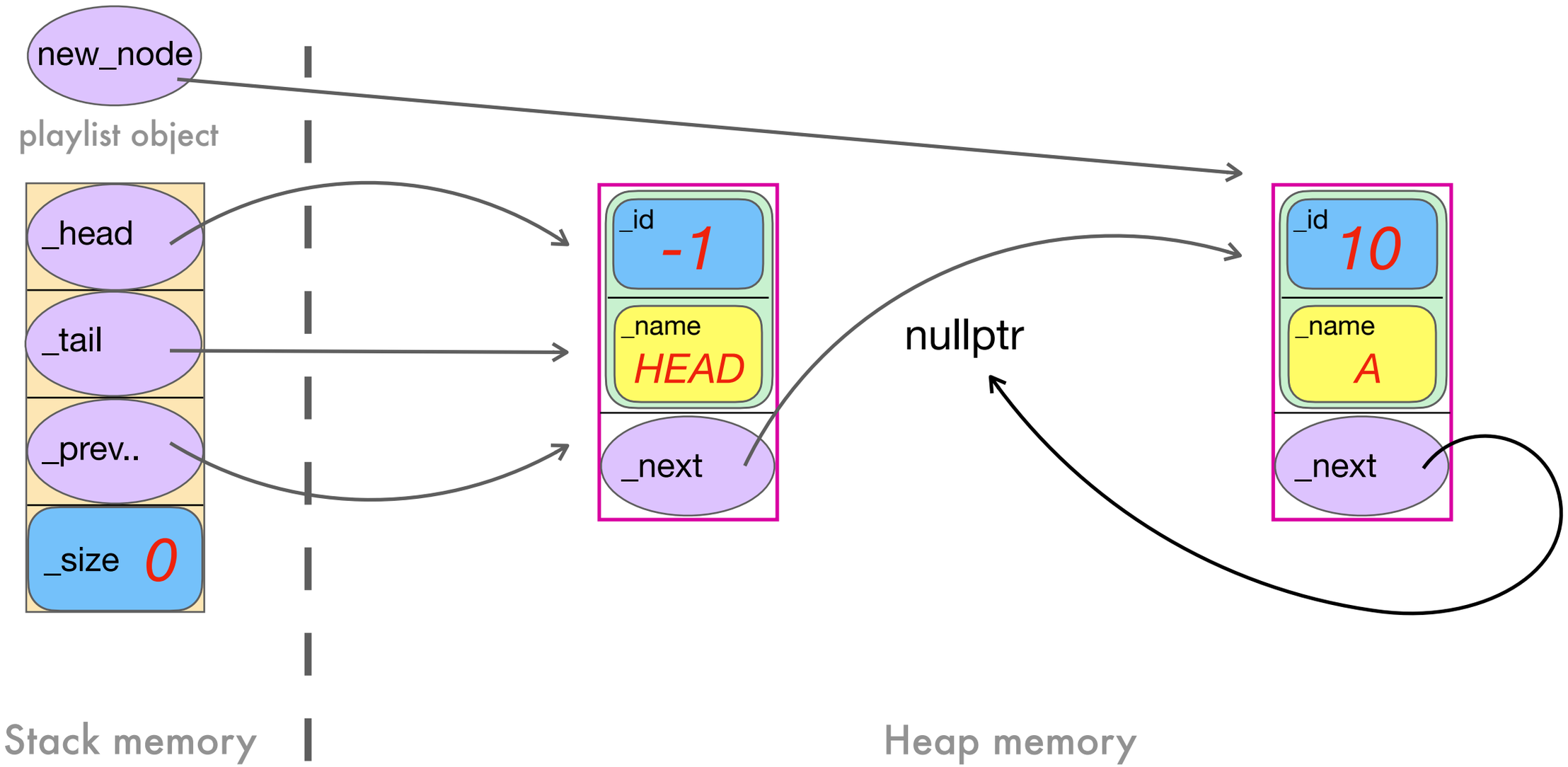


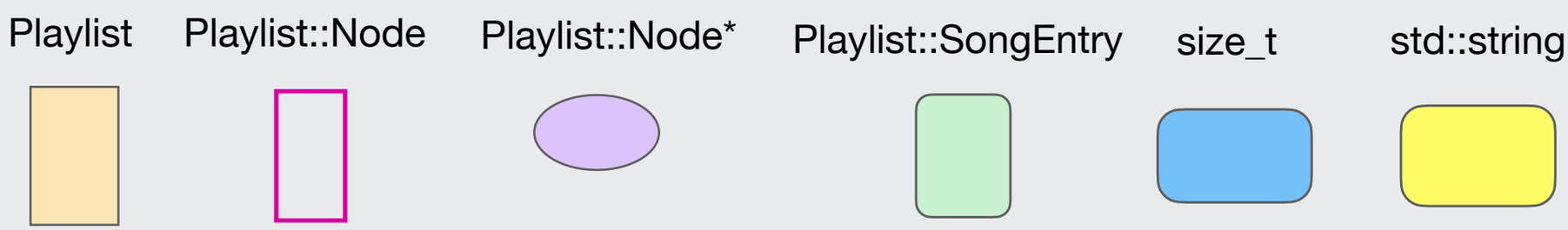
..and link it into the list.





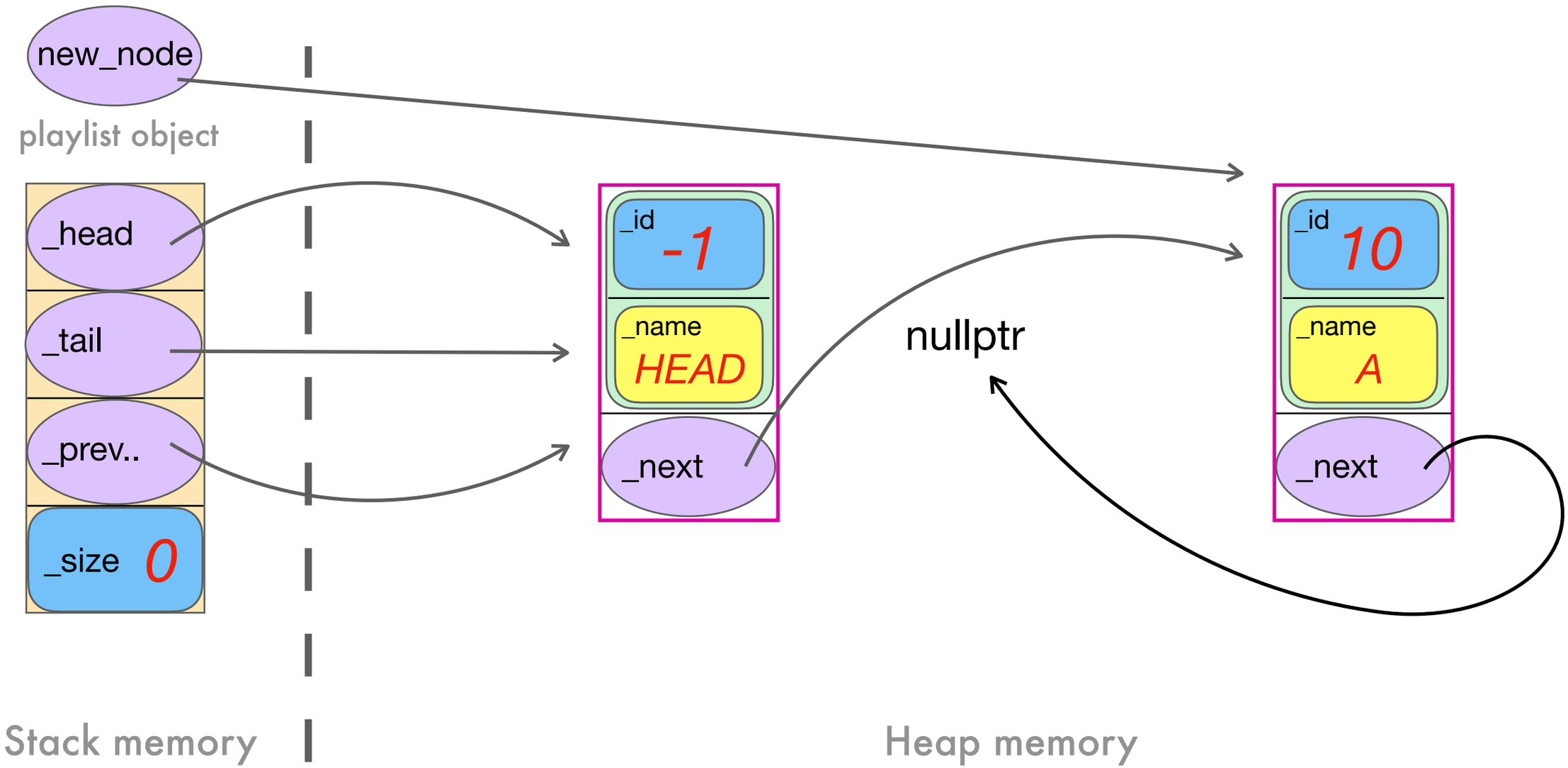
..and link it into the list.

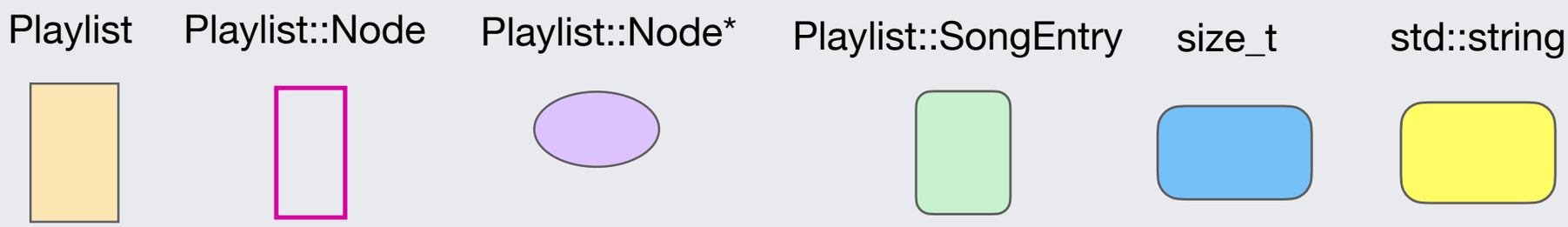




Legend

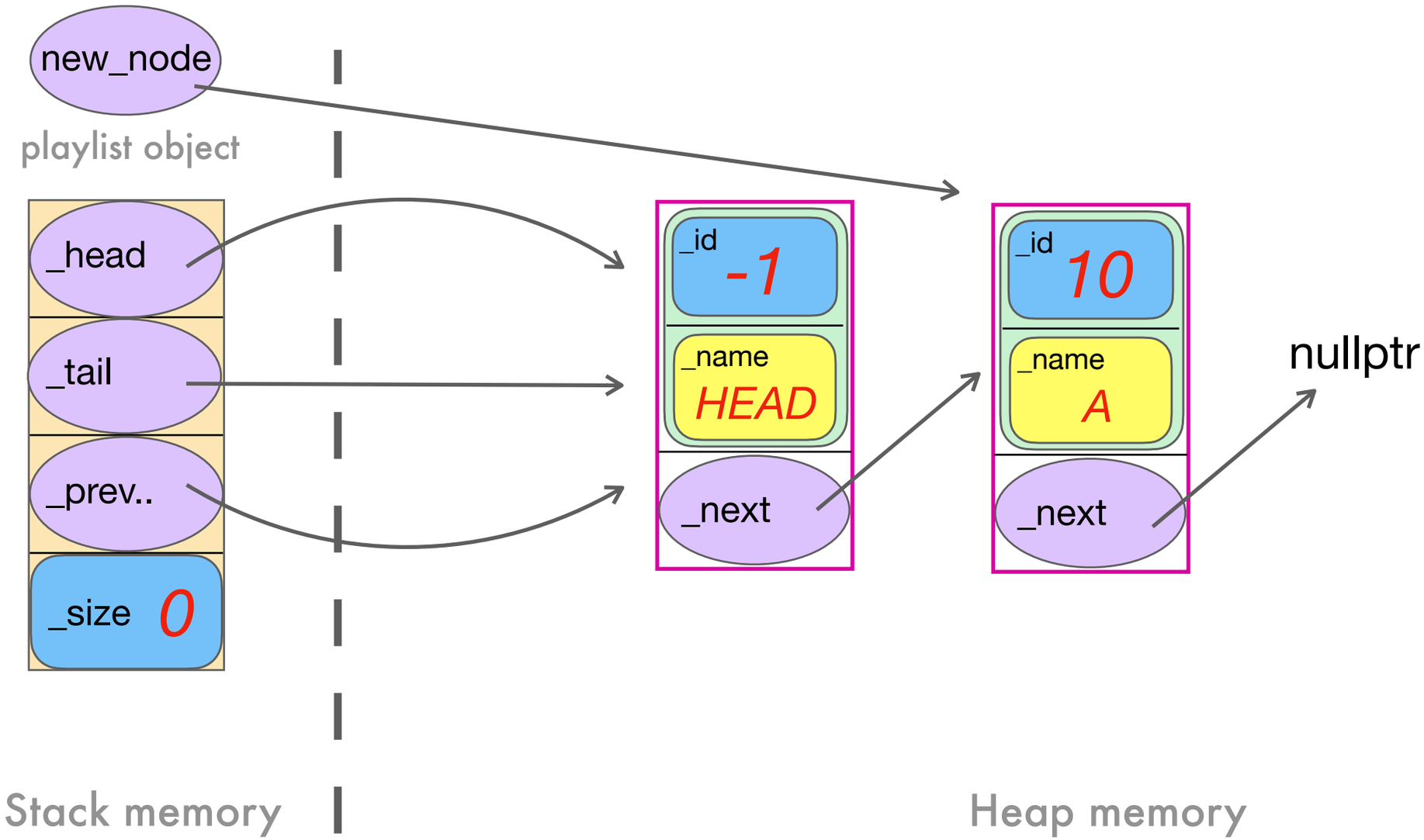
Woa. Let's clean up that picture.

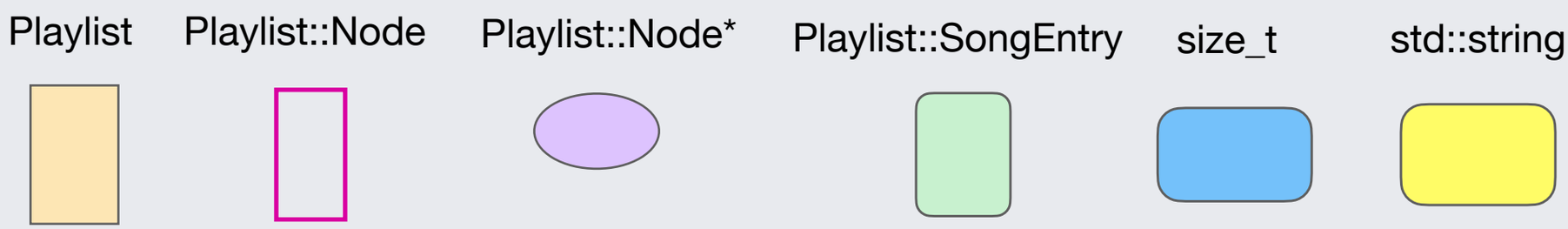




Legend

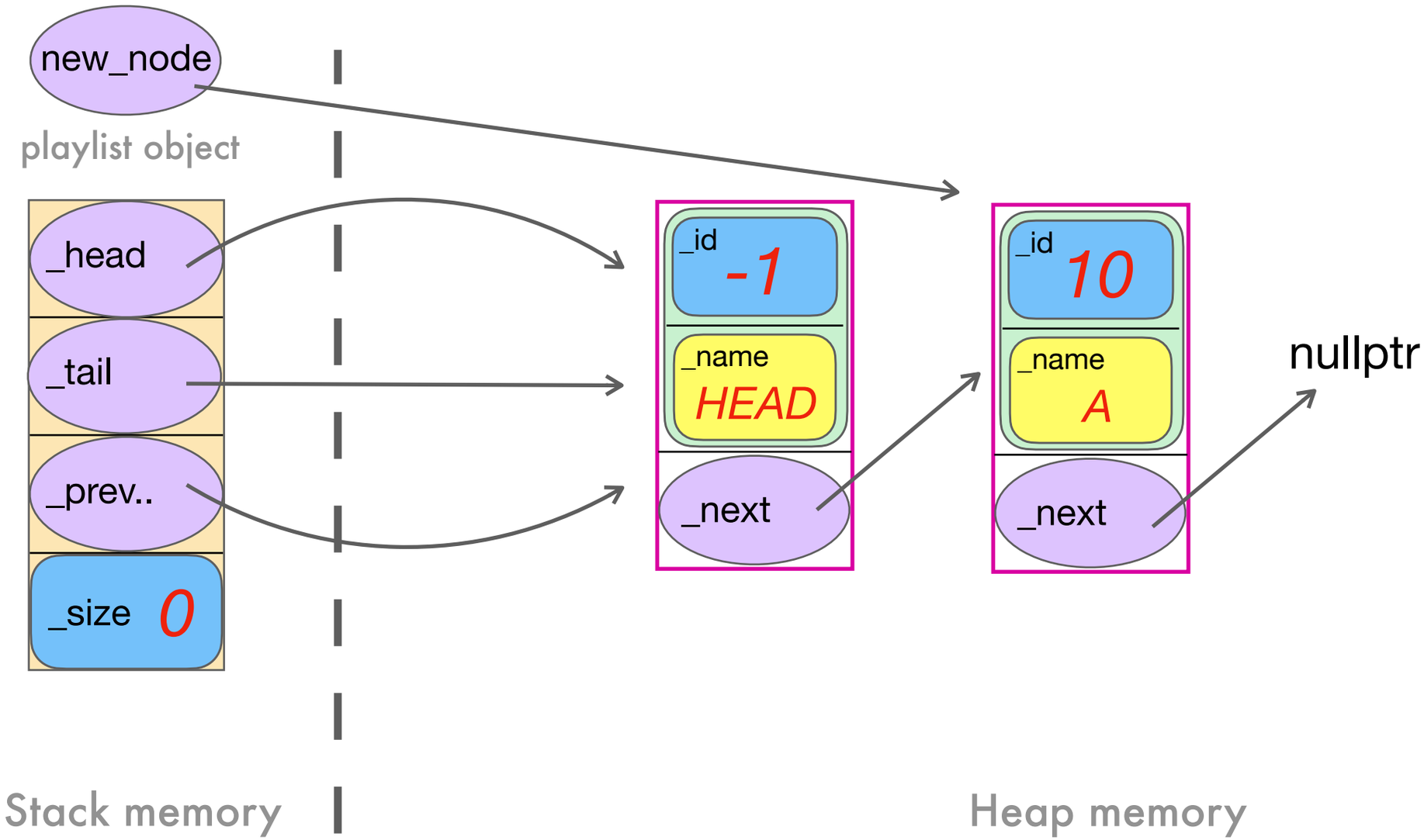
Woa. Let's clean up that picture.

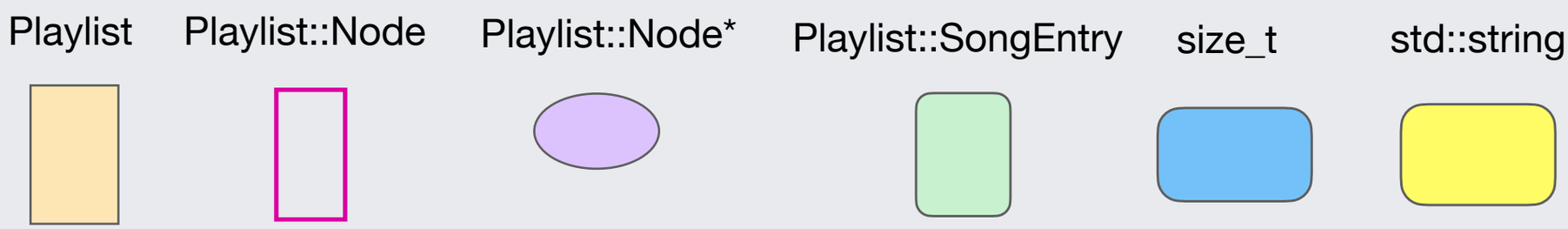




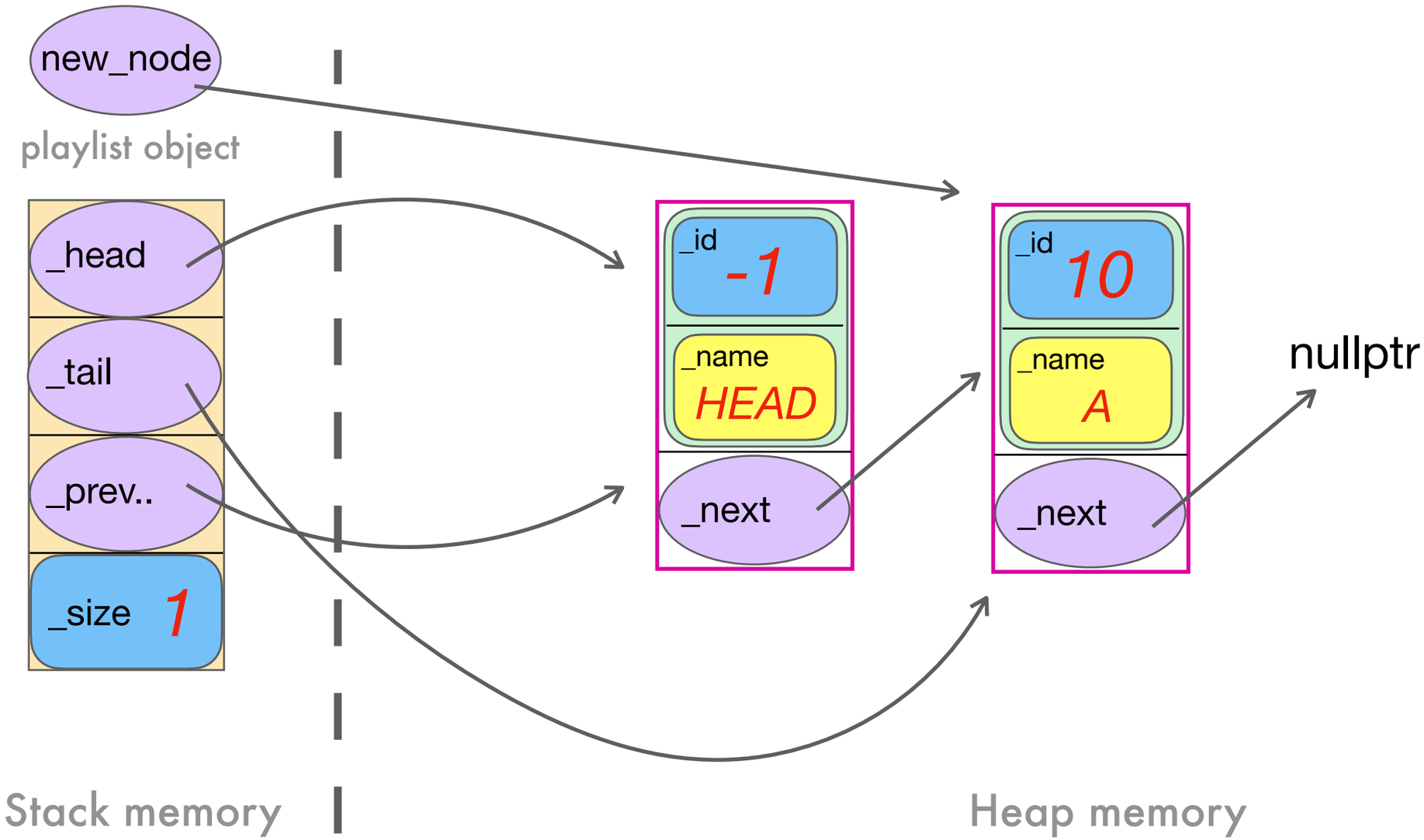
Legend

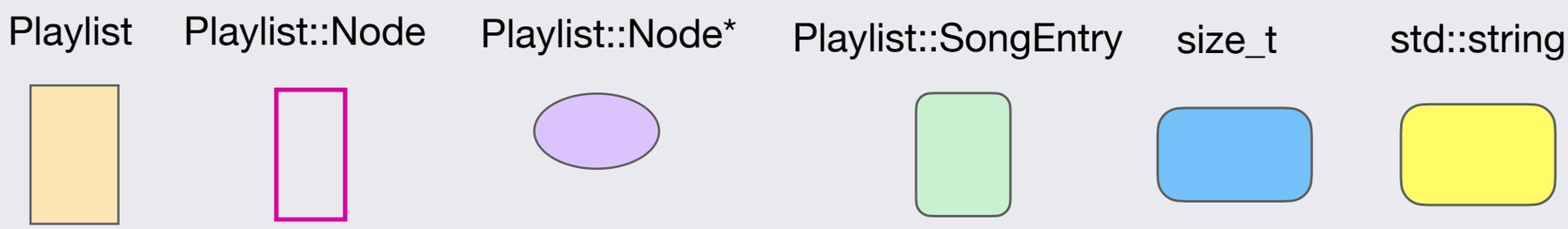
insert_at_cursor() needs to maintain Playlist data members. In this case, we need to update _tail and _size.



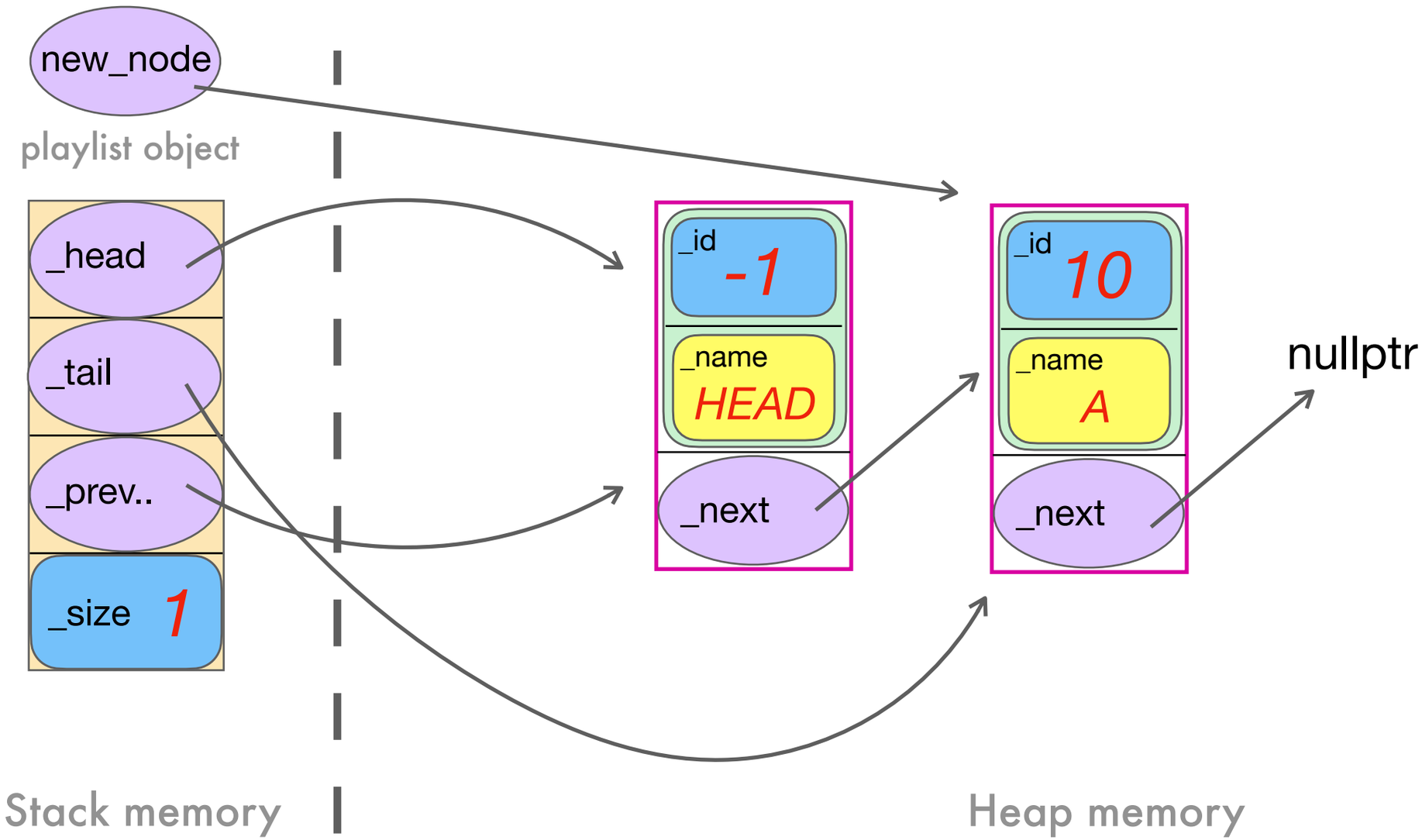


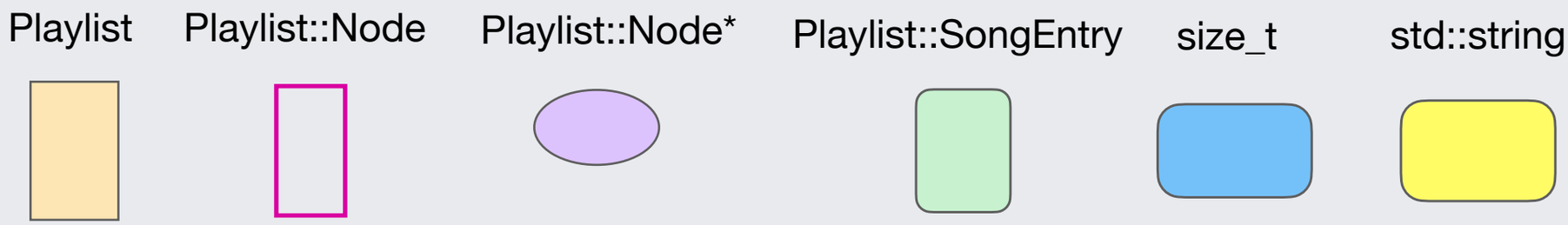
insert_at_cursor() needs to maintain Playlist data members. In this case, we need to update _tail and _size.



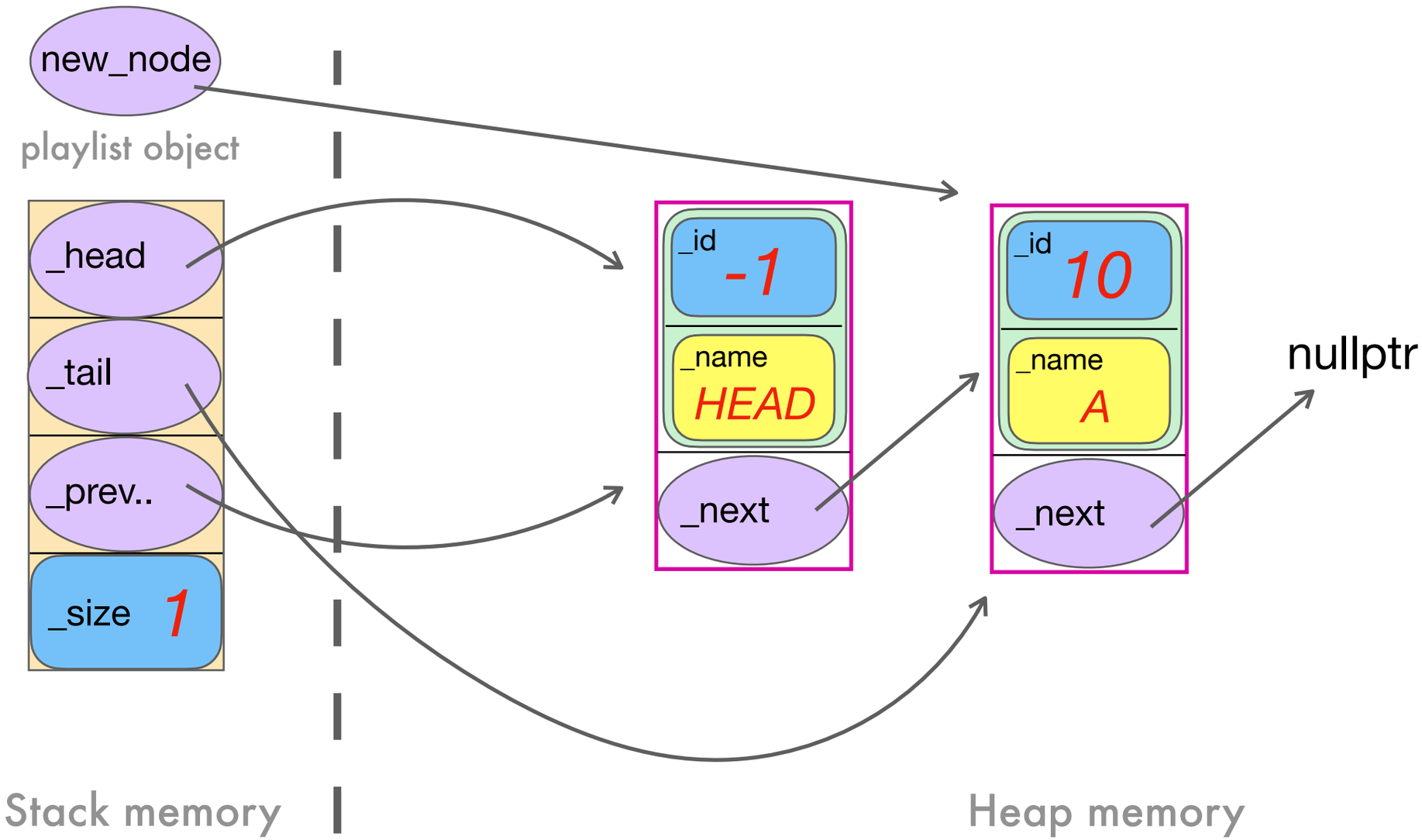


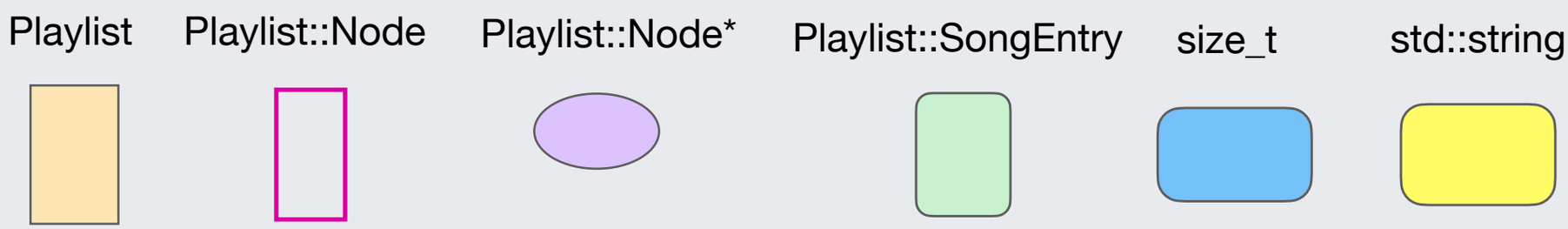
Neither _head nor _previous_to_current move. Indeed, _head will never move. It always points at the head sentinel node and is our way in to the list.





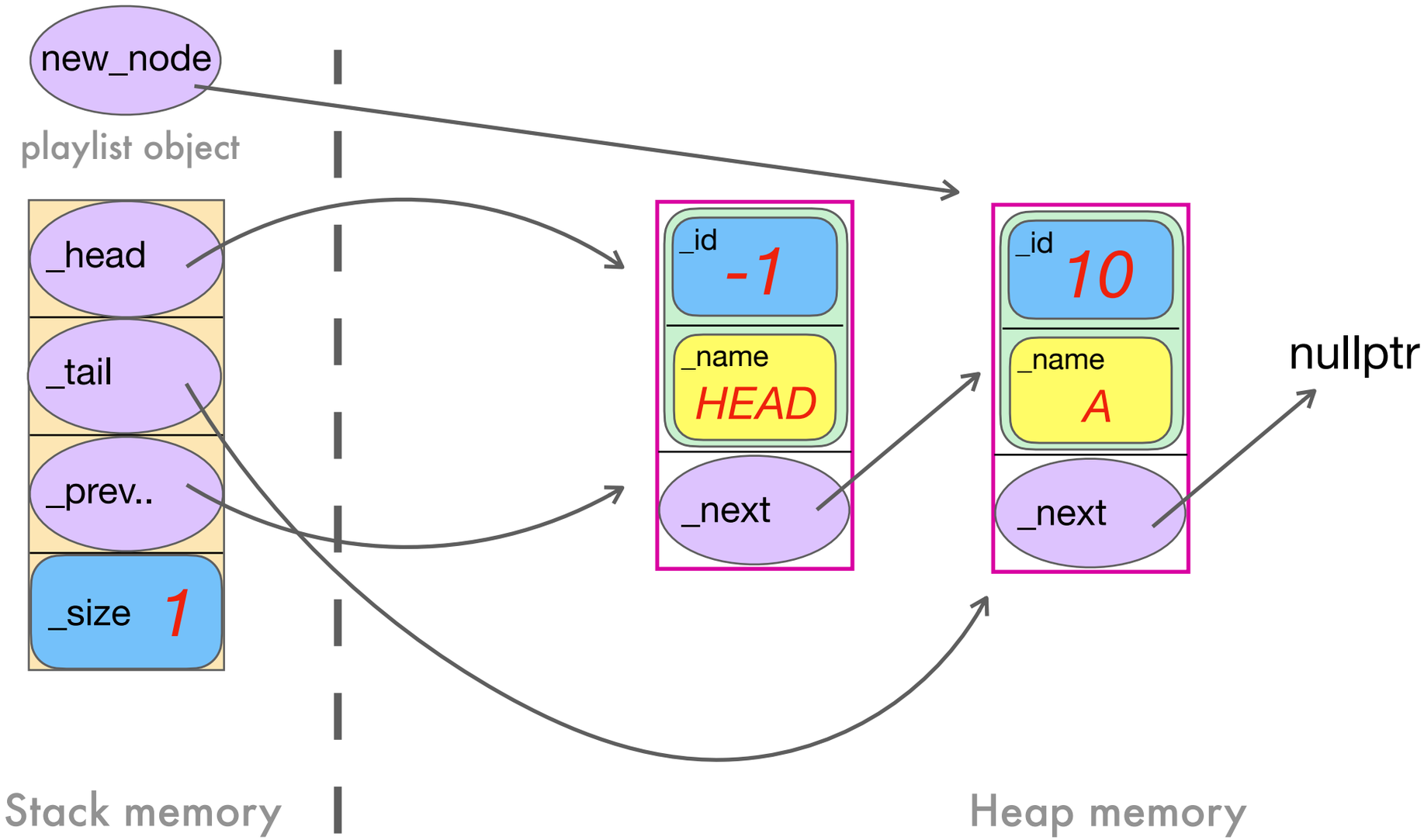
When insert_at_cursor() returns, the temporary local Node* new_node will be destroyed, but the node itself in heap memory is now accessible via the list, via the sentinel node.

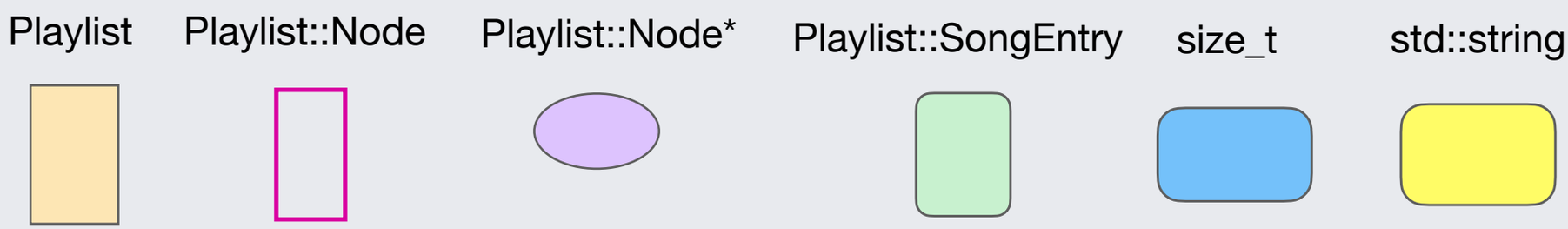




Legend

When insert_at_cursor() returns, the temporary local Node* new_node will be destroyed, but the node itself in heap memory is now accessible via the list, via the sentinel node.

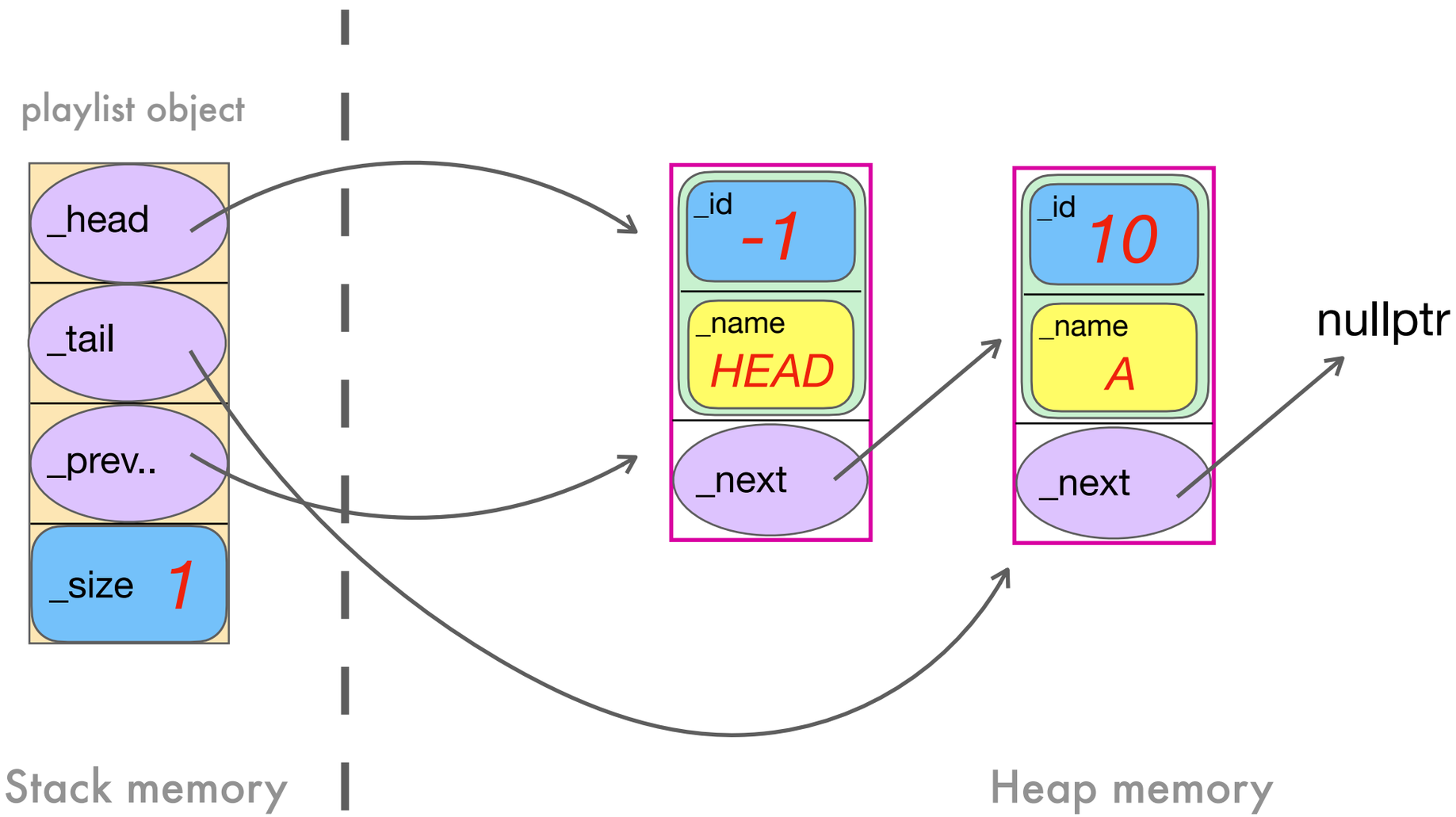


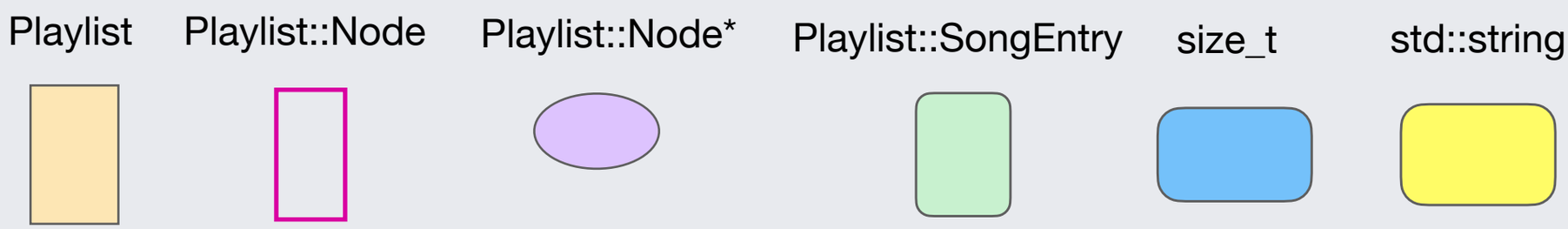


Legend

Let’s insert another song:

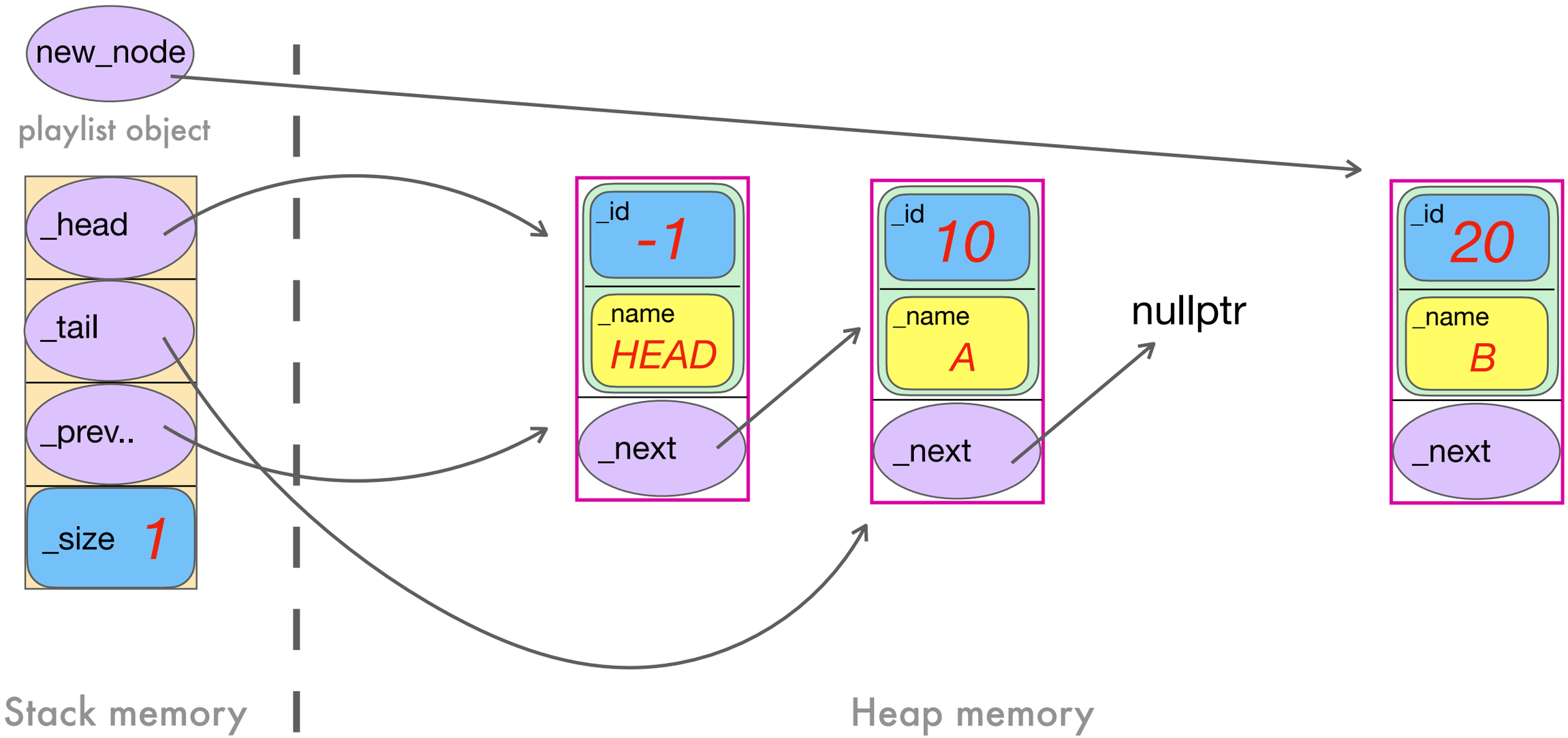
```
playlist.insert_at_cursor(Playlist::SongEntry(20, “B”));
```

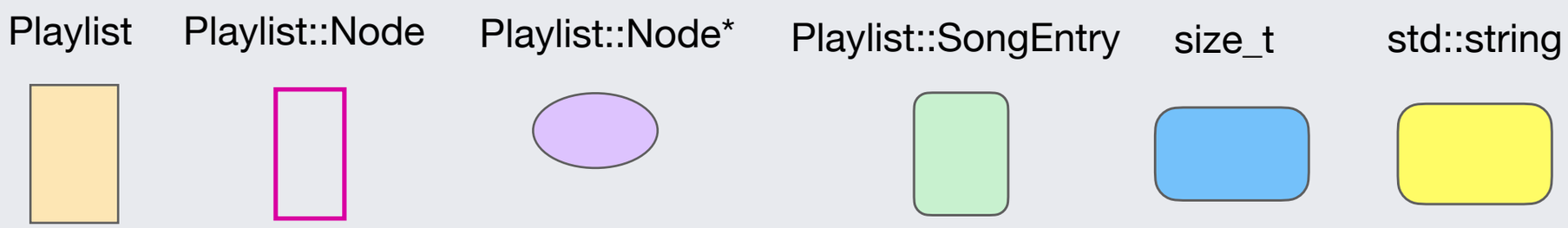




Let’s insert another song:

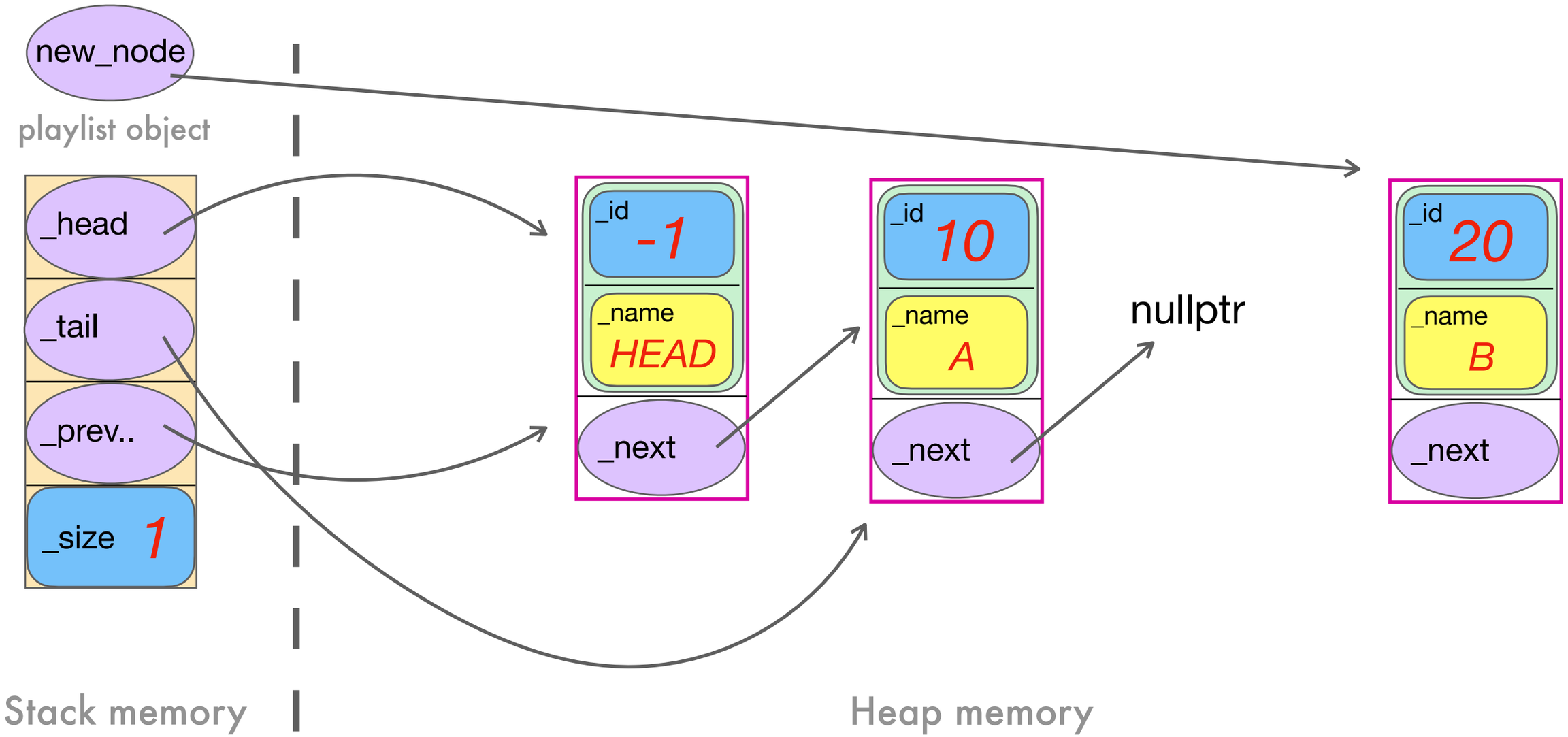
```
playlist.insert_at_cursor(Playlist::SongEntry(20, “B”));
```

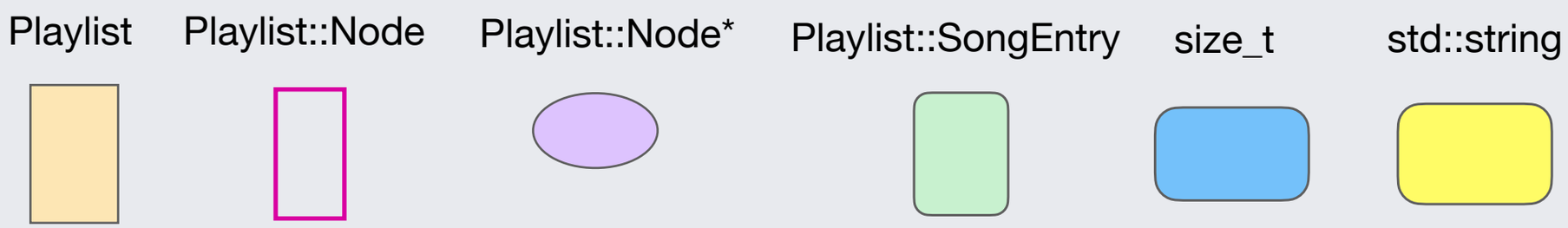




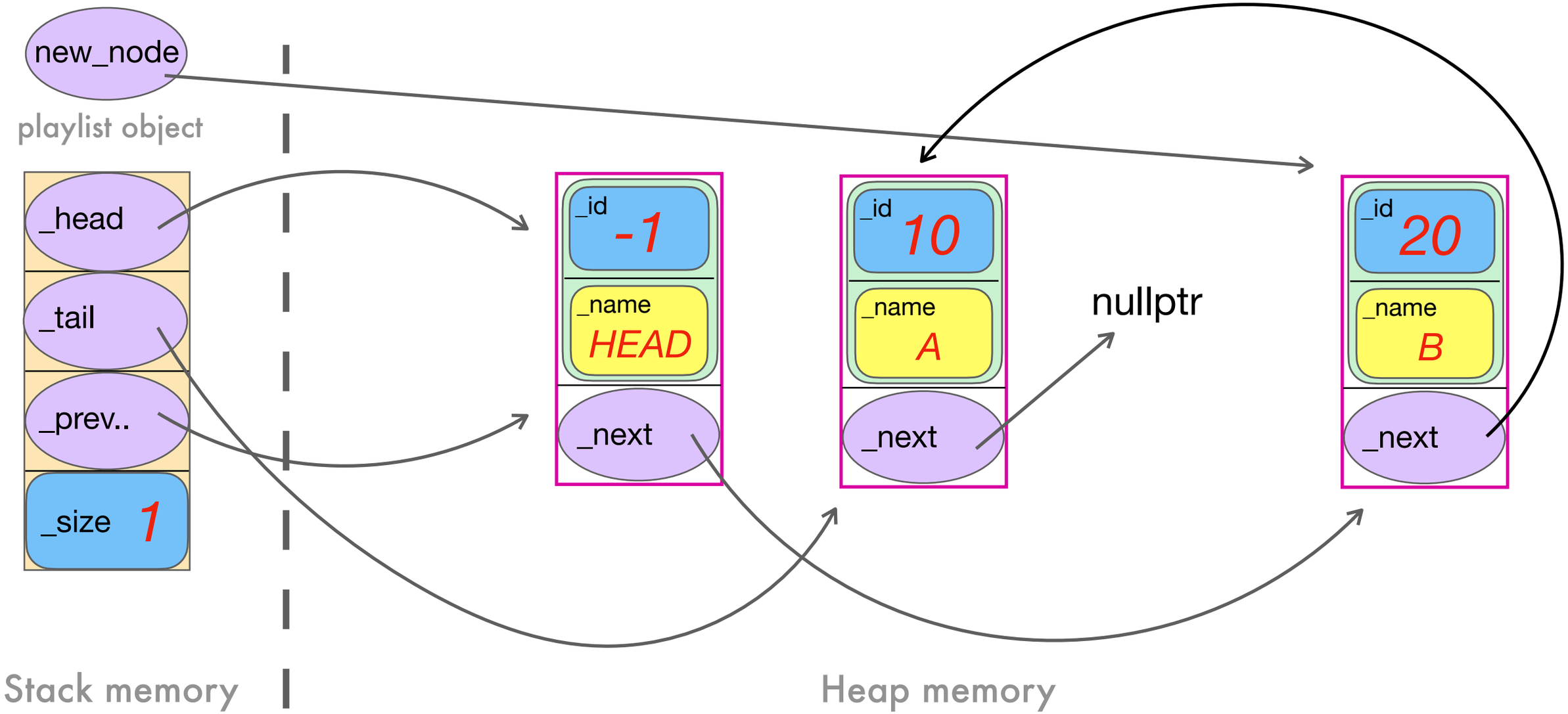
Legend

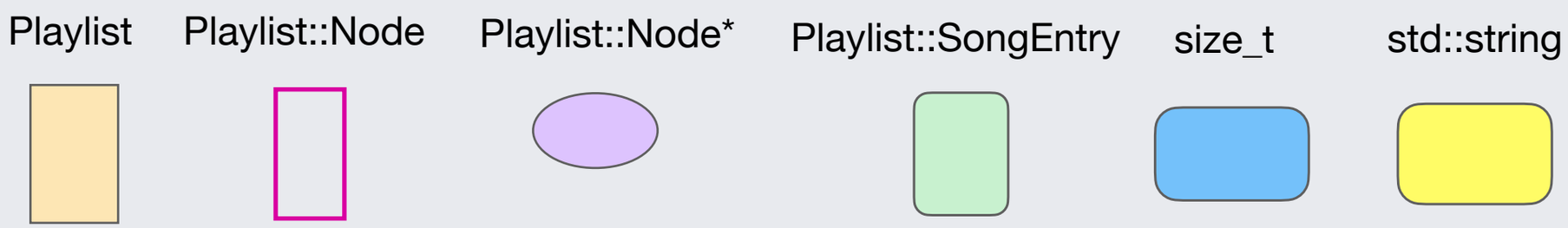
We're inserting after _prev_to_current which right now points to the sentinel node. So we need to carefully re-assign _next pointers..



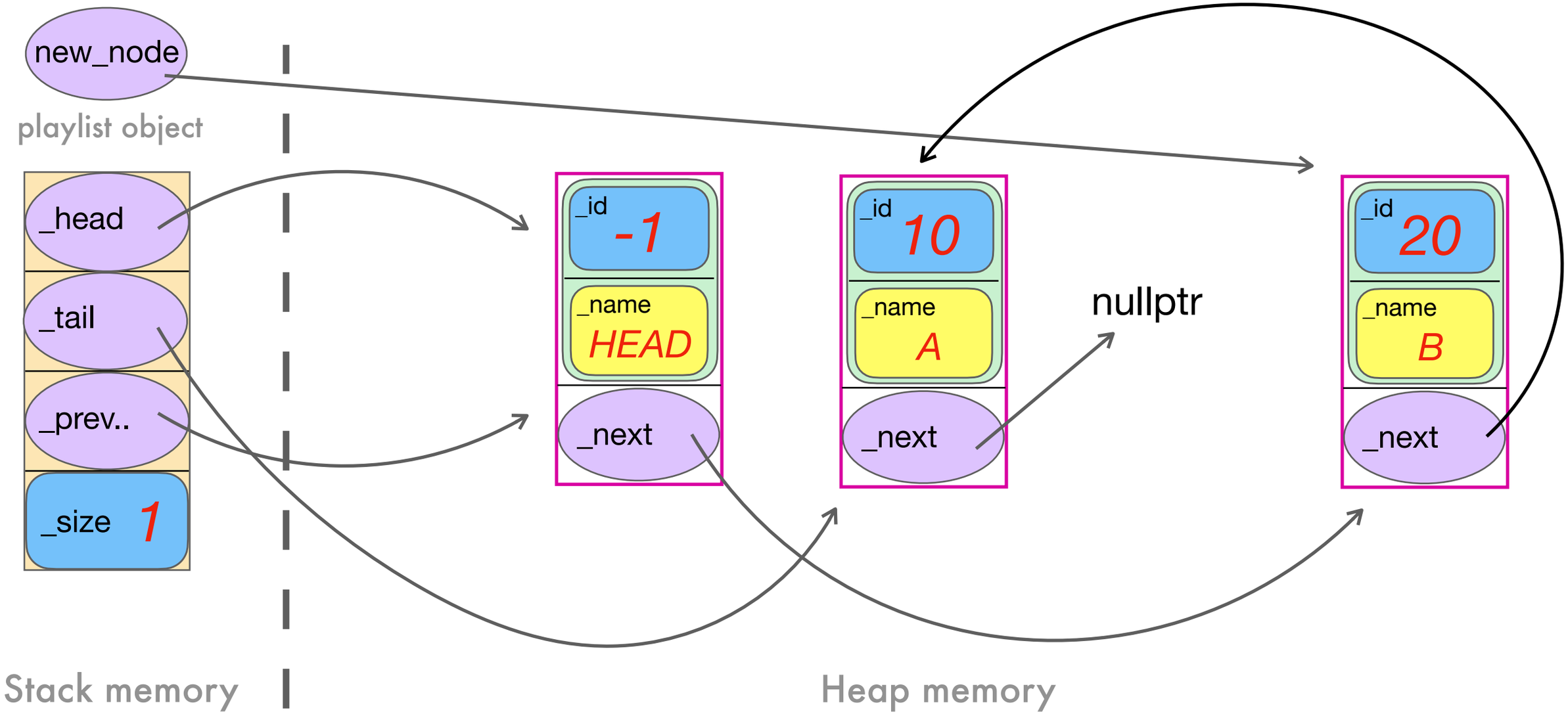


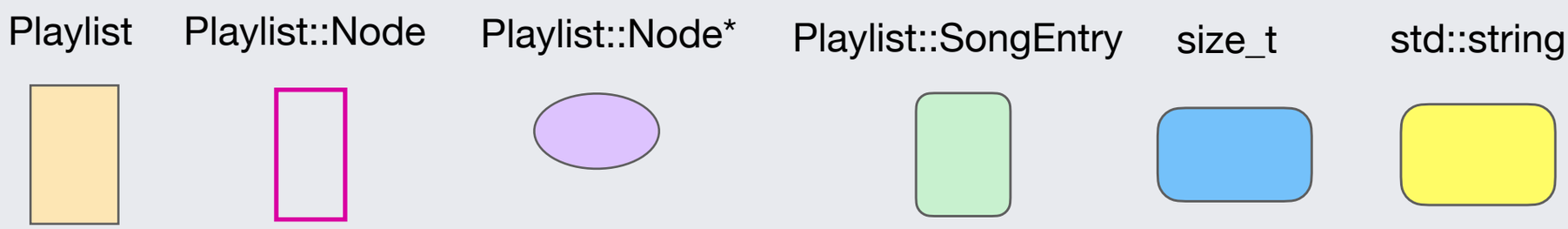
We're inserting after _prev_to_current which right now points to the sentinel node. So we need to carefully re-assign _next pointers..





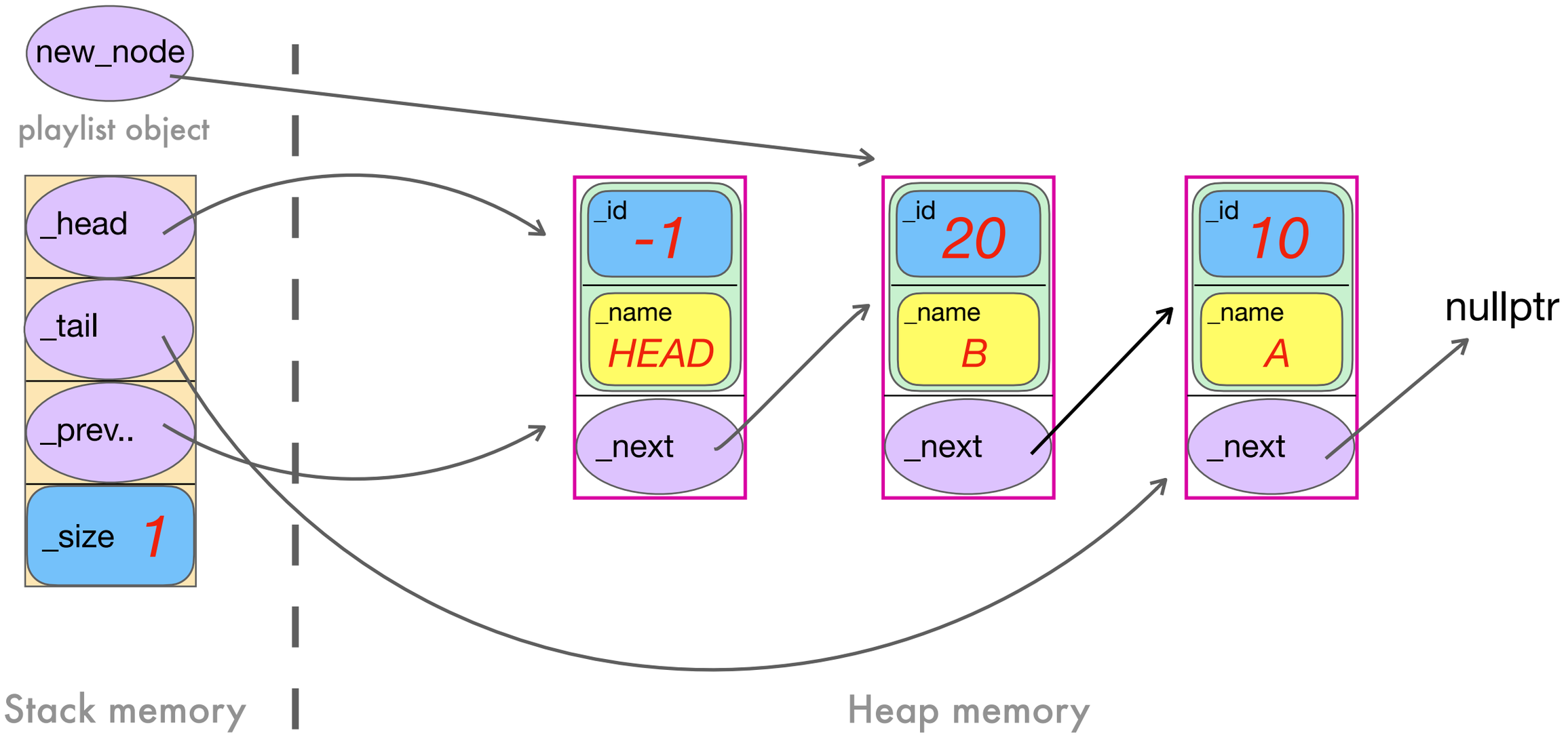
Woa. Let's clean up that picture.

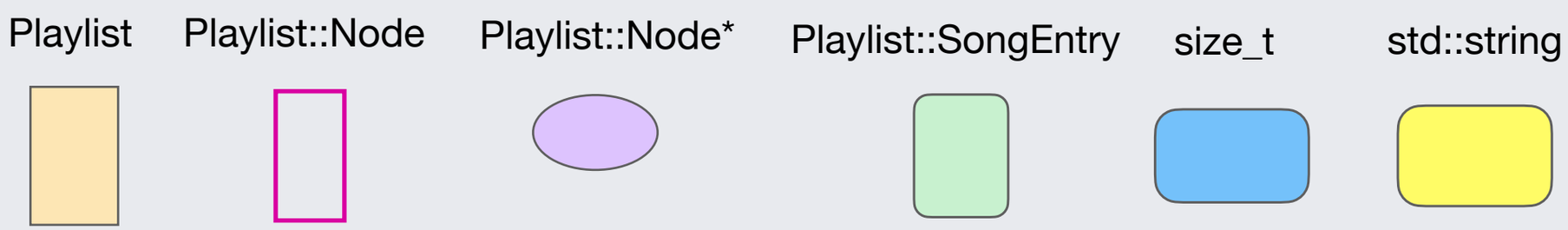




Legend

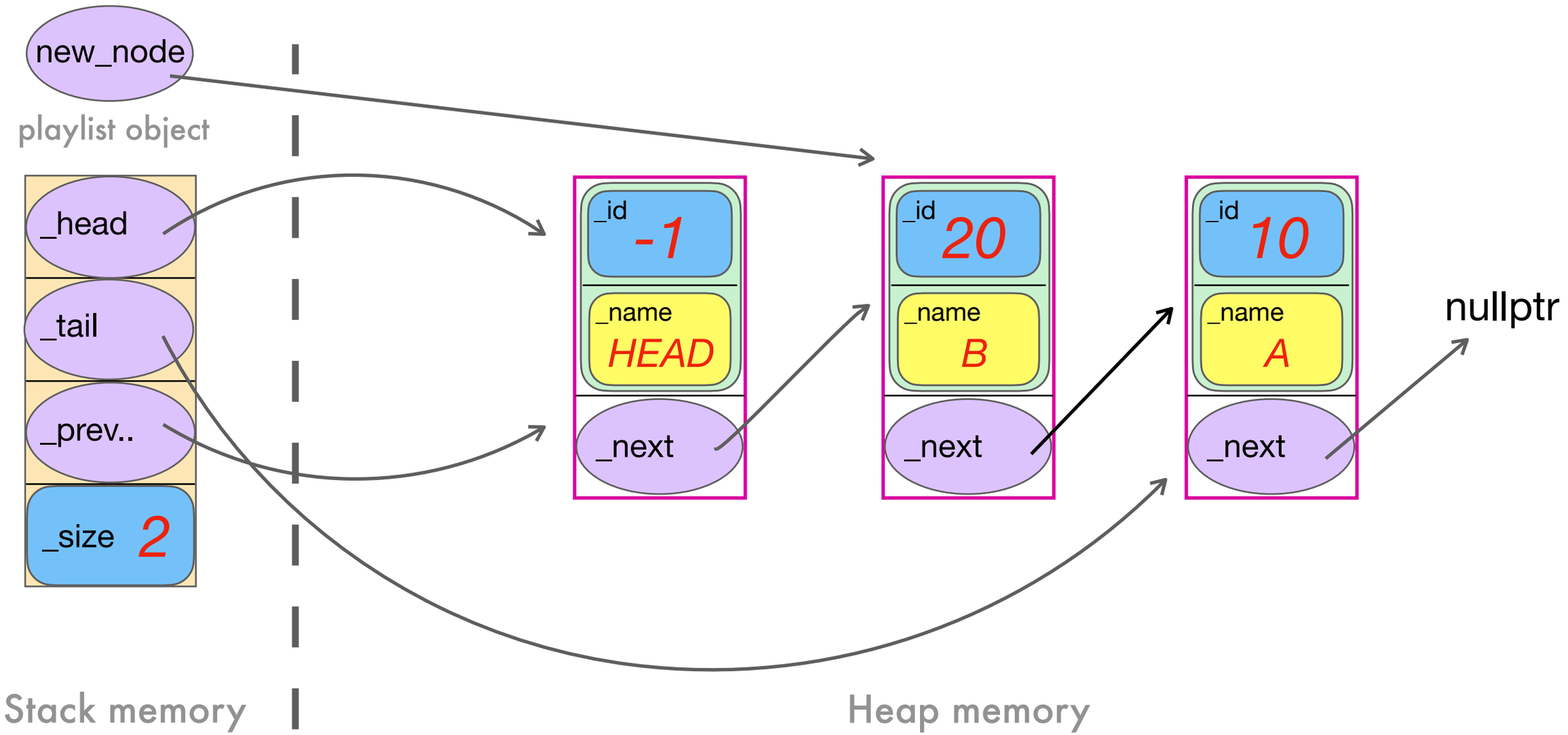
Note that in this case, _tail does not need to move, but we need to update the size before the function returns.

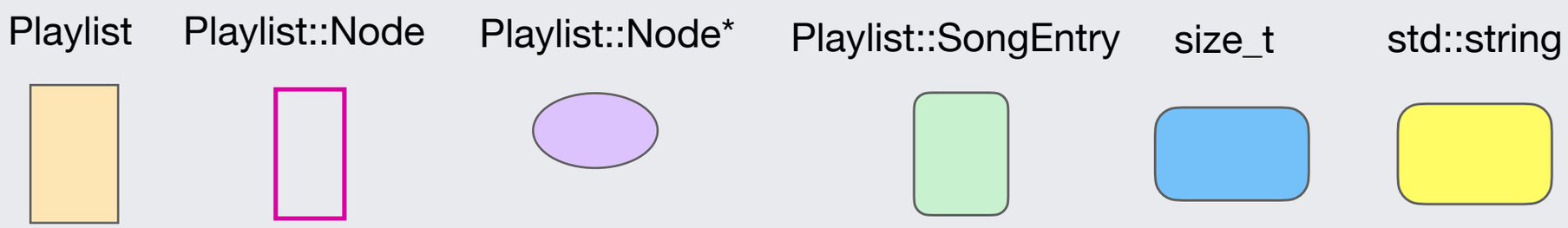




Legend

Note that in this case, _tail does not need to move, but we need to update the size before the function returns.

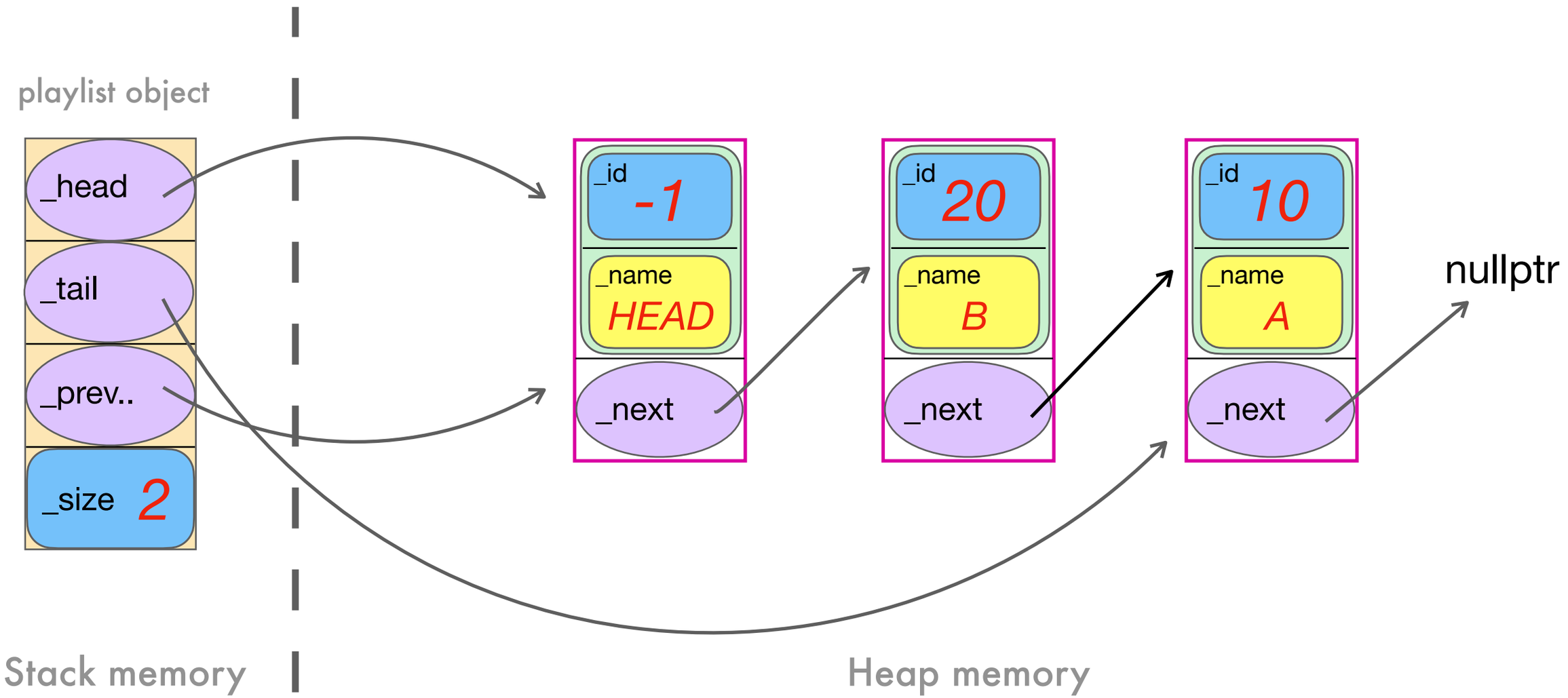


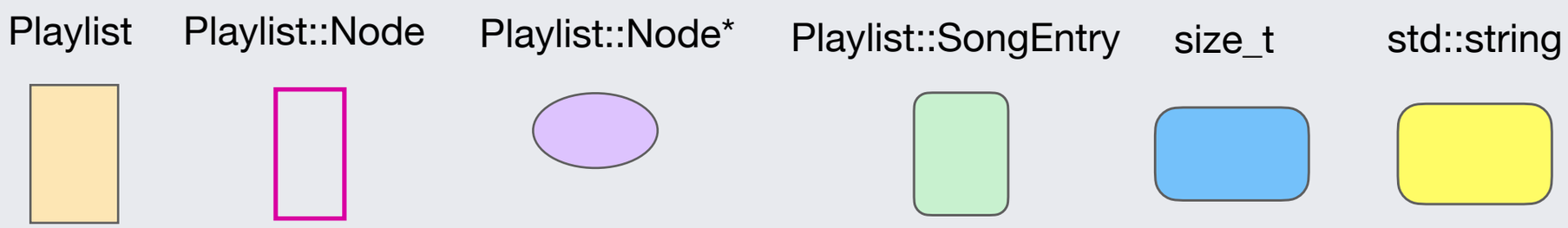


Legend

Now let's advance the cursor:

```
playlist.advance_cursor();
```

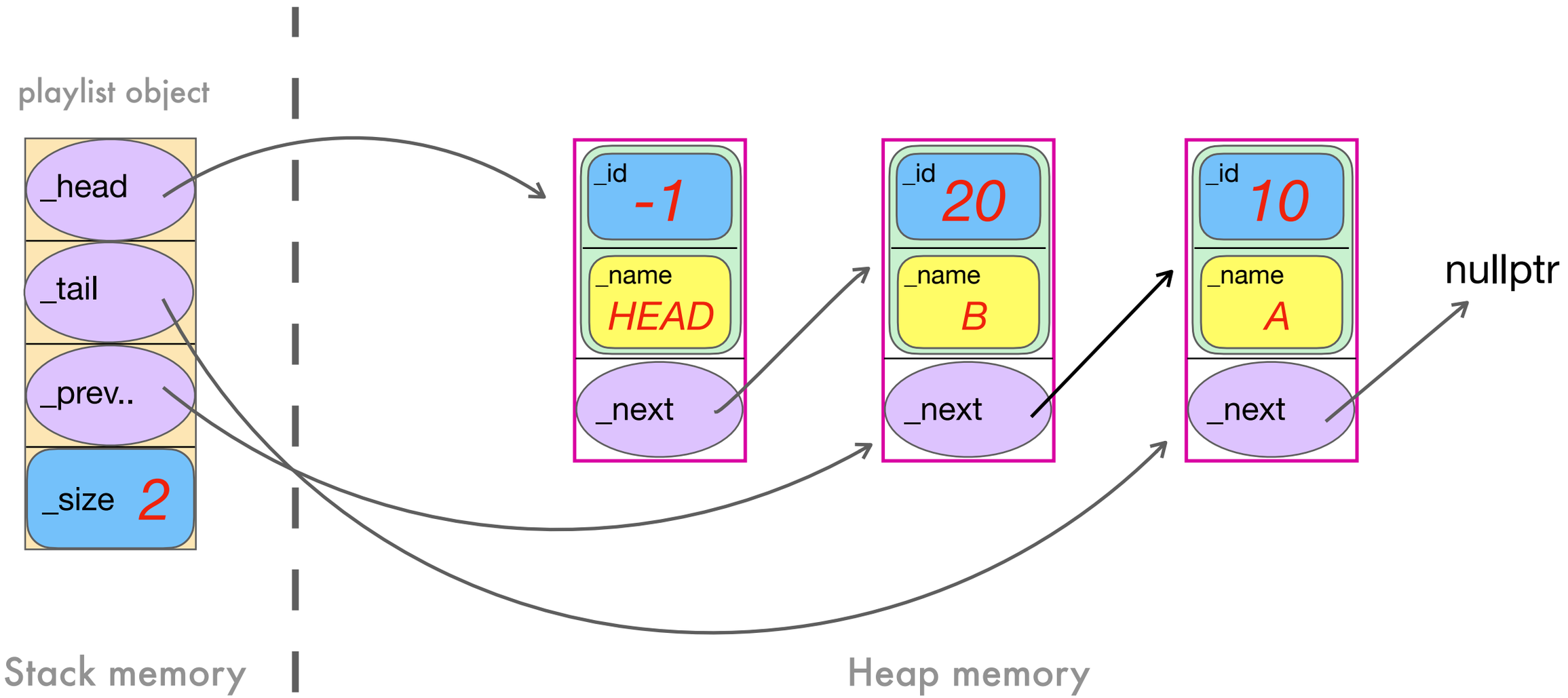


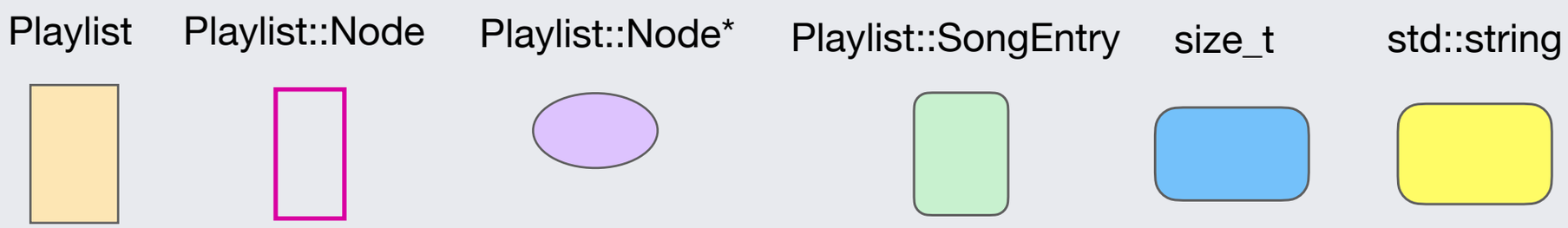


Legend

Now let's advance the cursor:

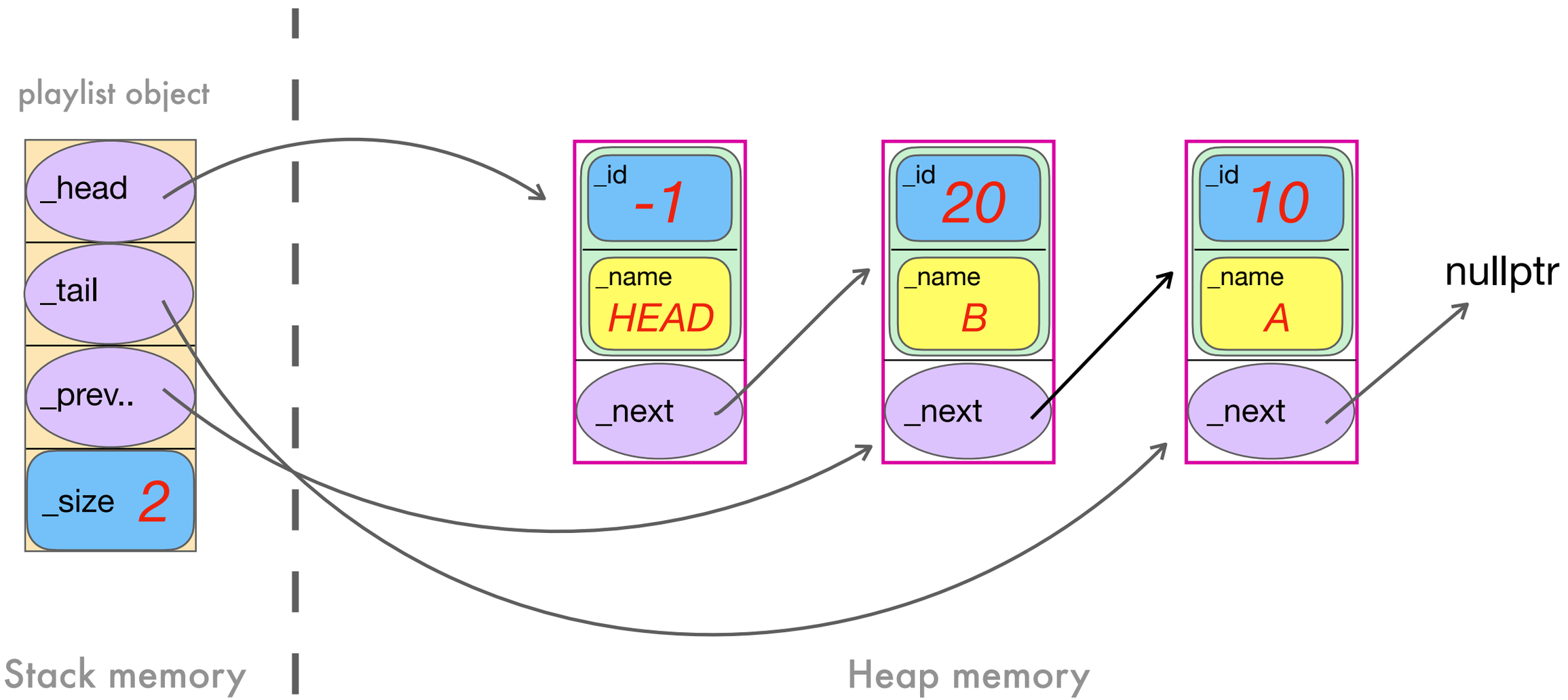
```
playlist.advance_cursor();
```

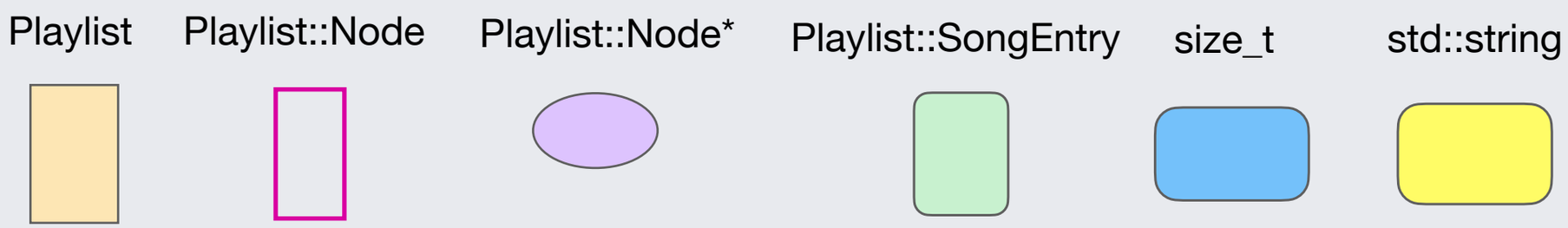




And insert once more:

```
playlist.insert_at_cursor(Playlist::SongEntry(30, “C”));
```

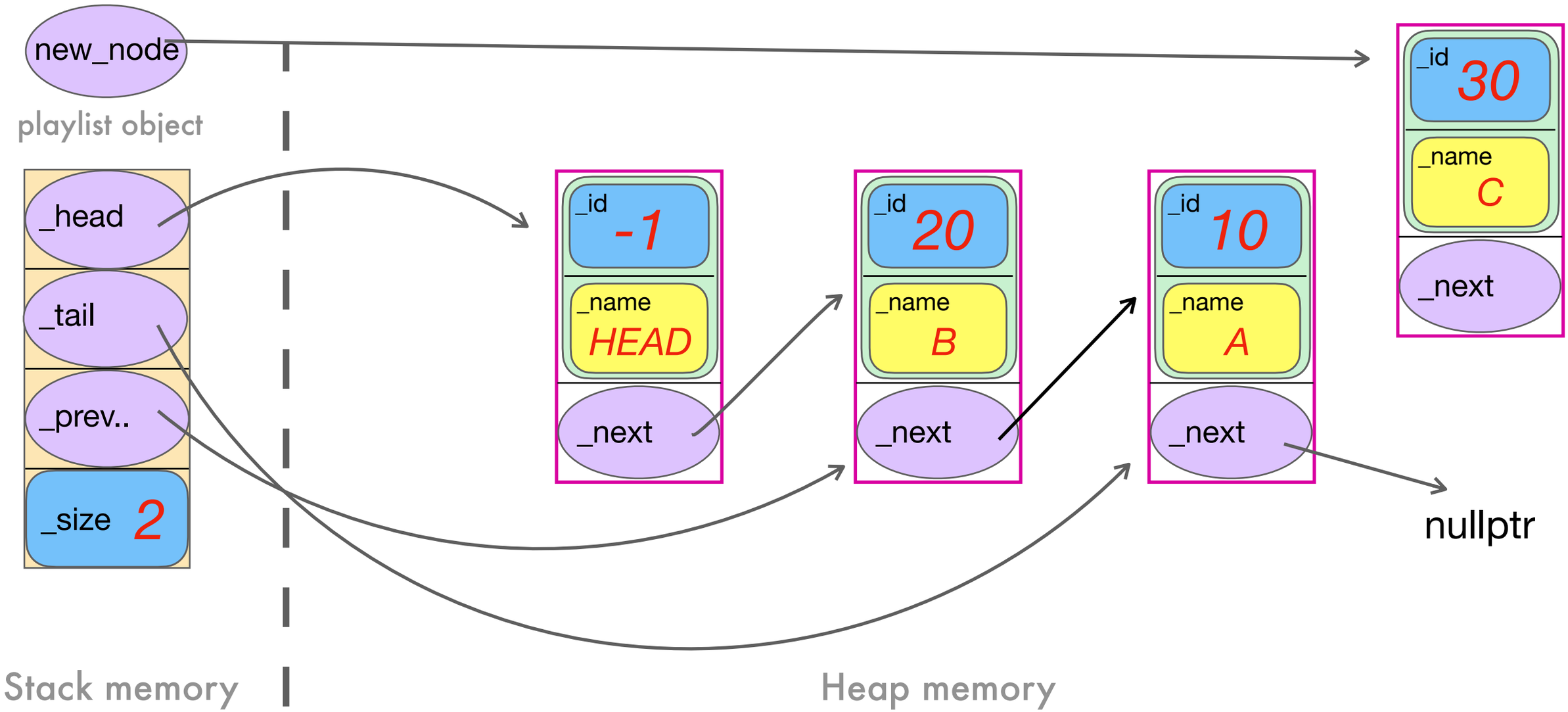


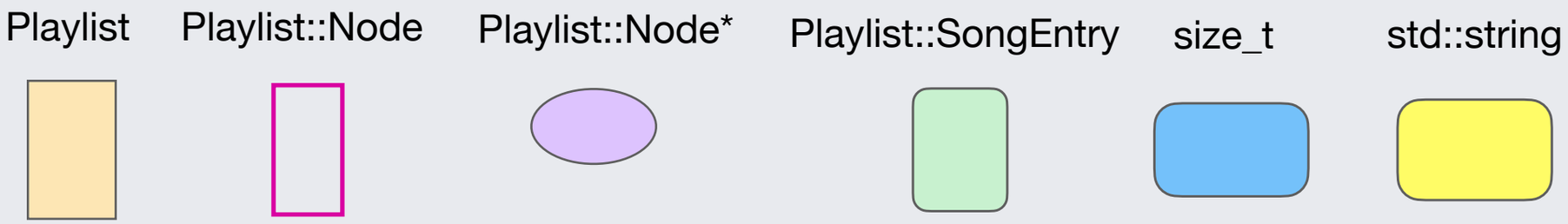


Legend

And insert once more:

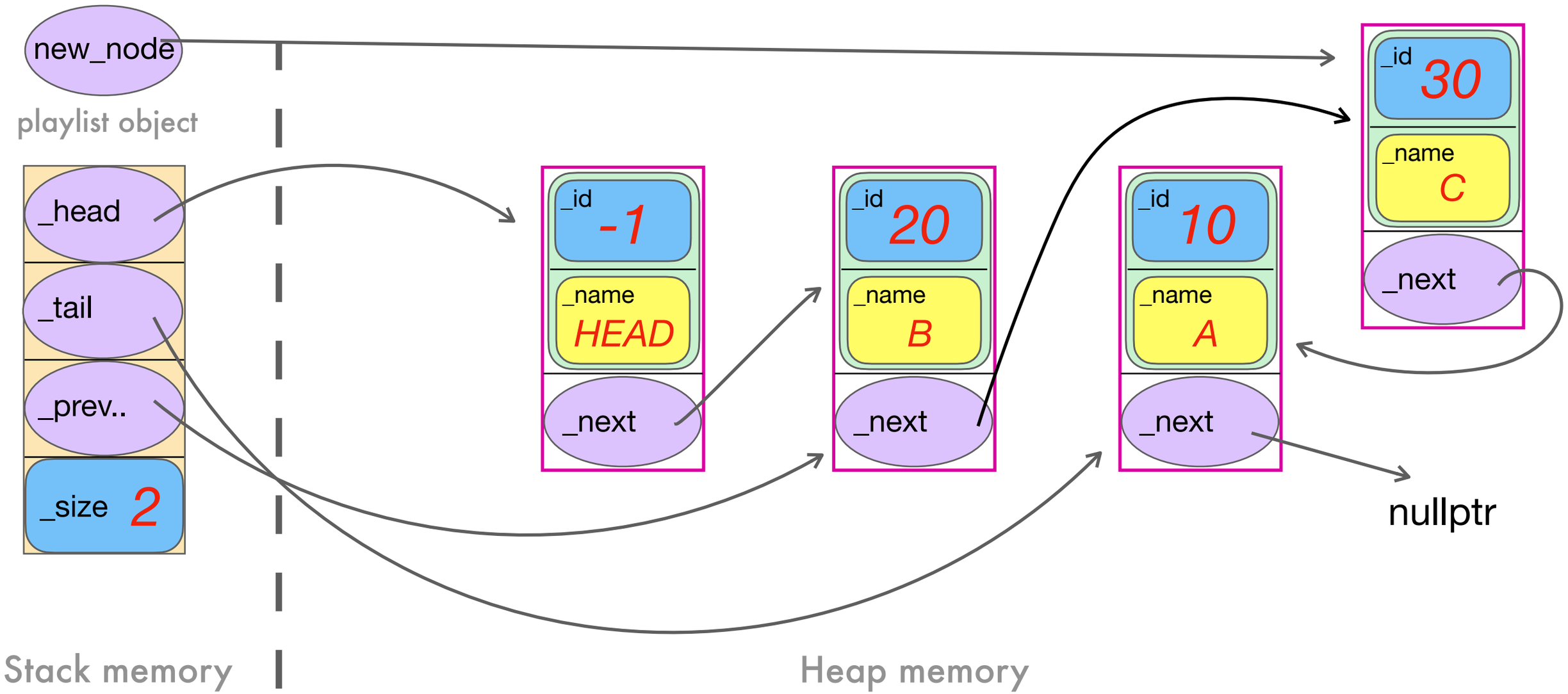
```
playlist.insert_at_cursor(Playlist::SongEntry(30, “C”);
```

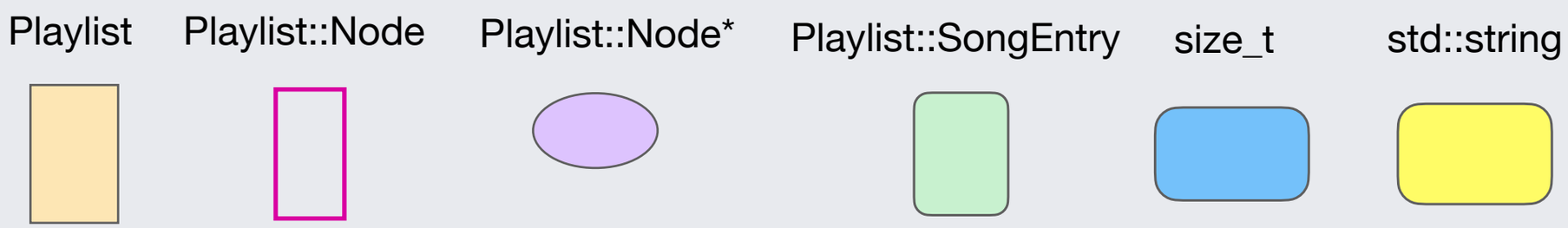




And insert once more:

```
playlist.insert_at_cursor(Playlist::SongEntry(30, “C”);
```

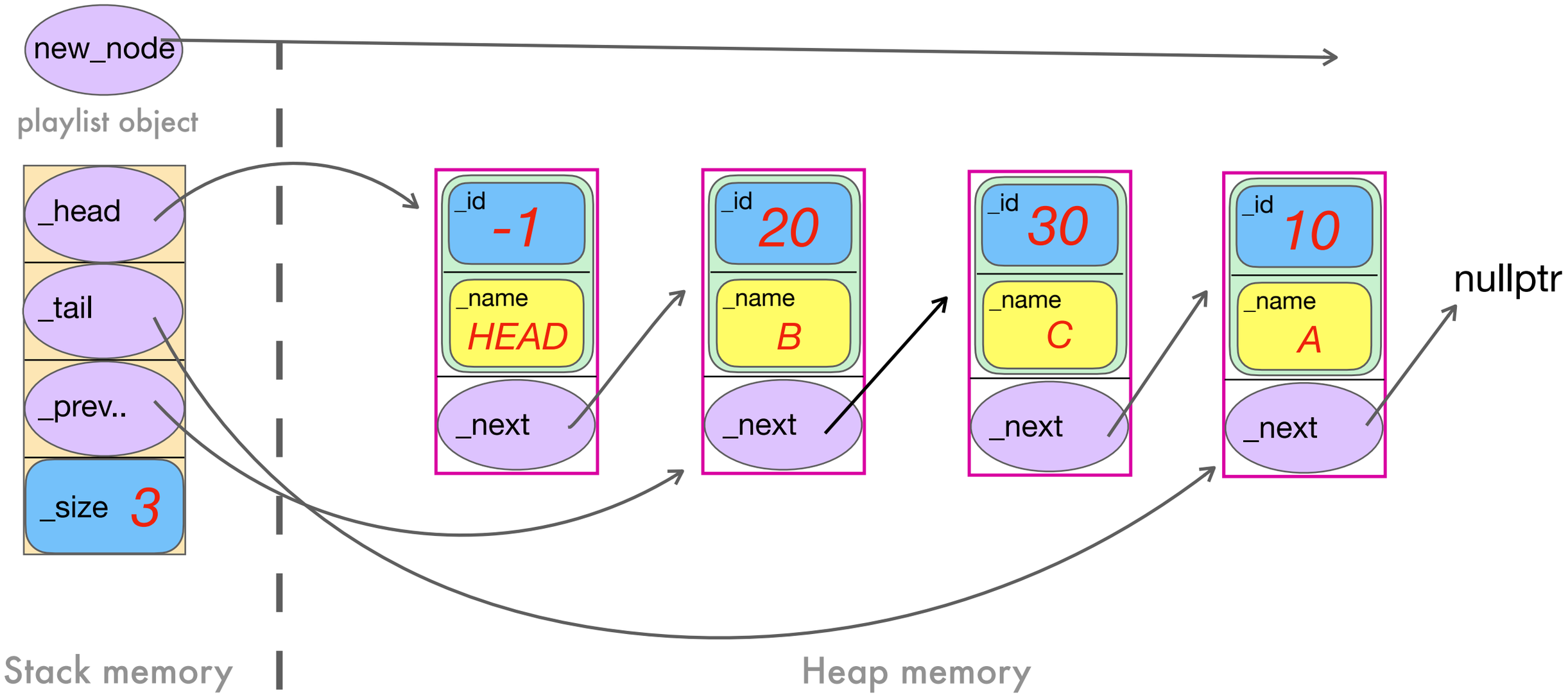


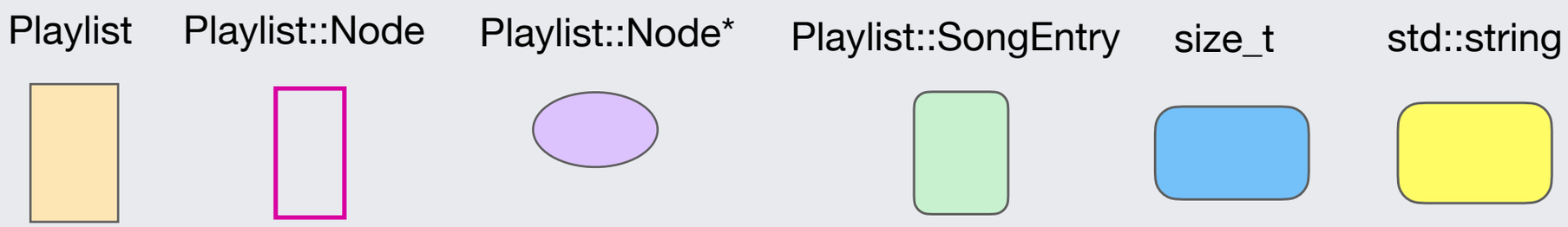


Legend

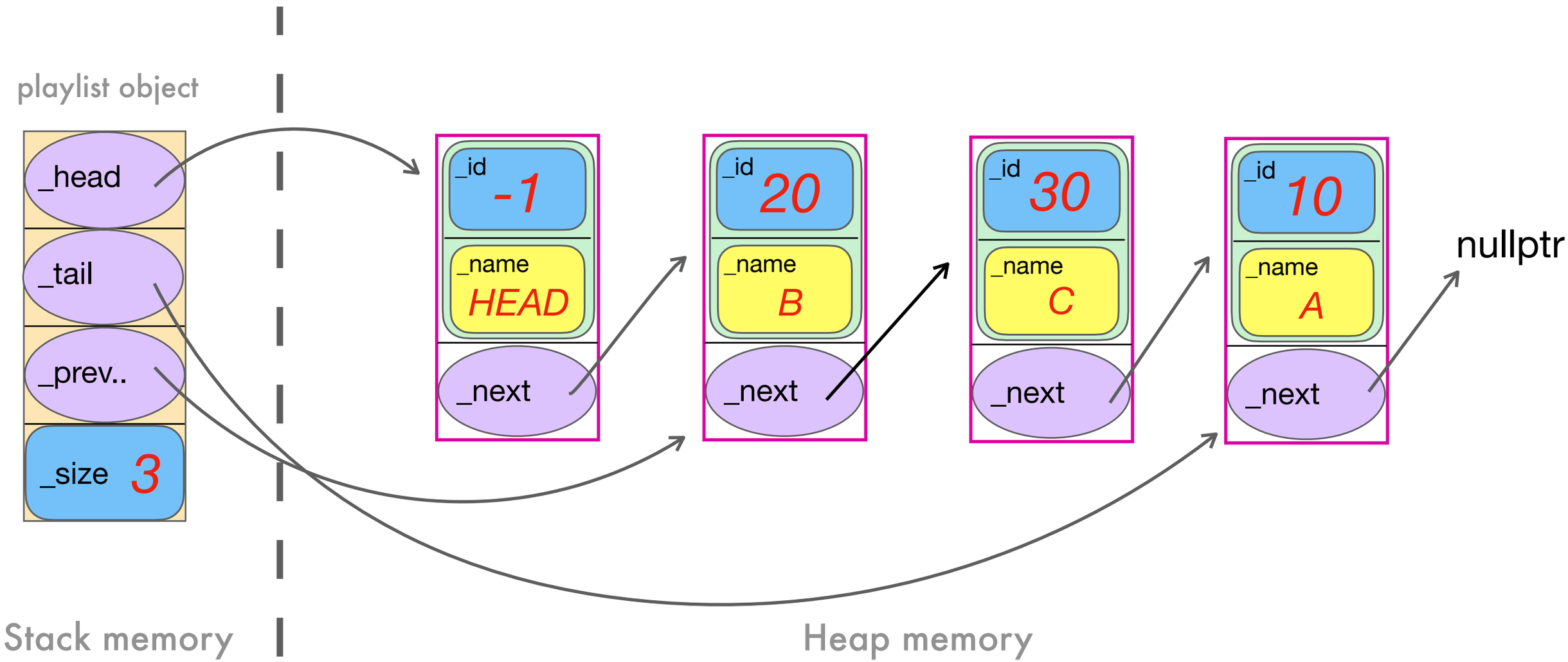
And insert once more:

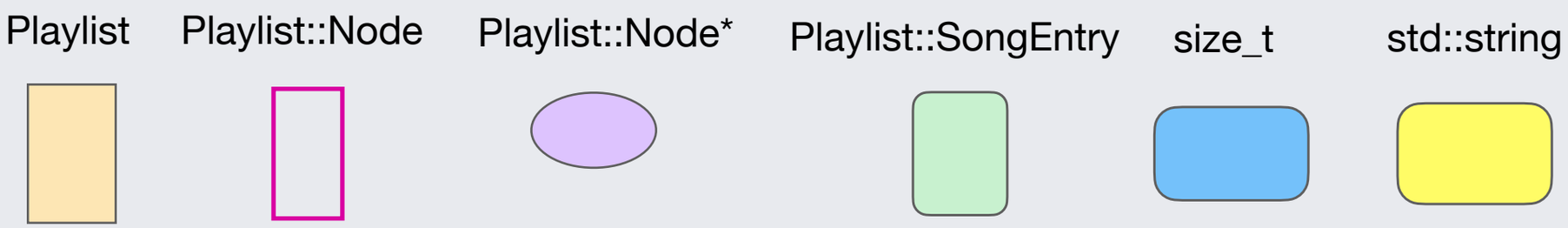
```
playlist.insert_at_cursor(Playlist::SongEntry(30, “C”);
```



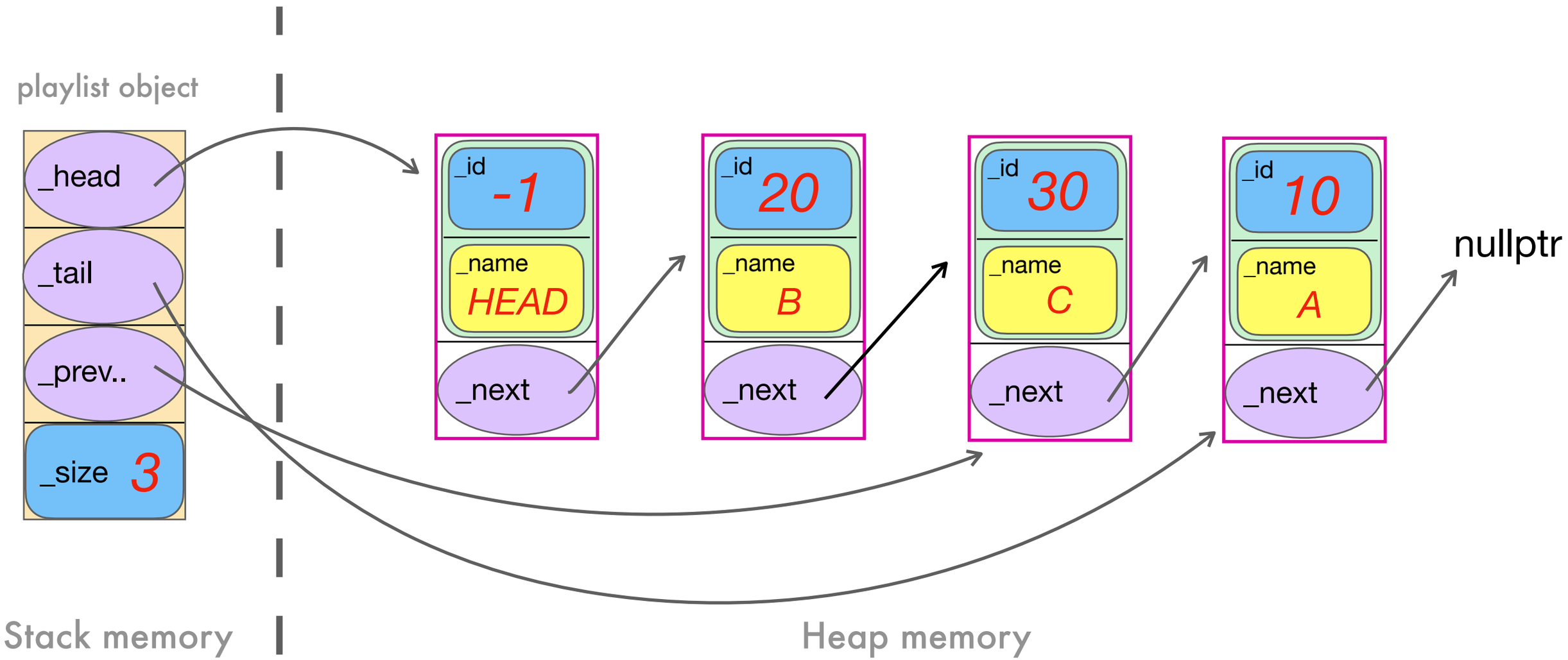


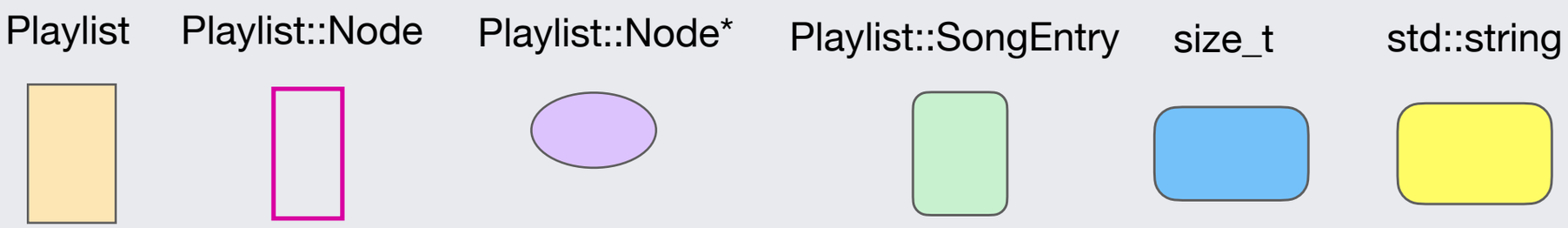
```
playlist.advance_cursor();
```



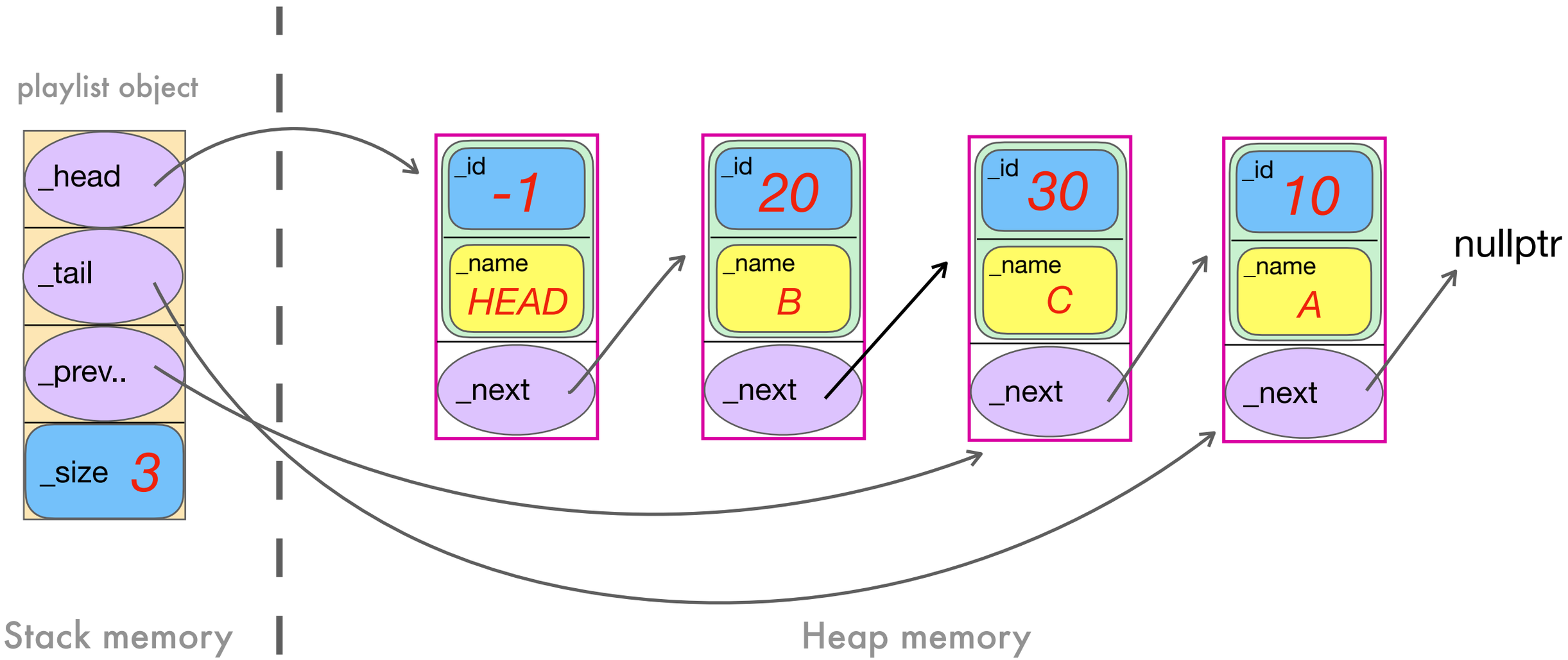


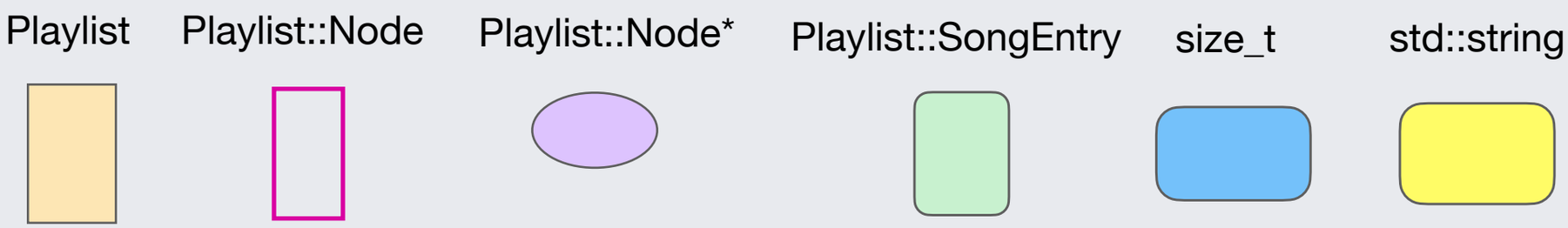
```
playlist.advance_cursor();
```



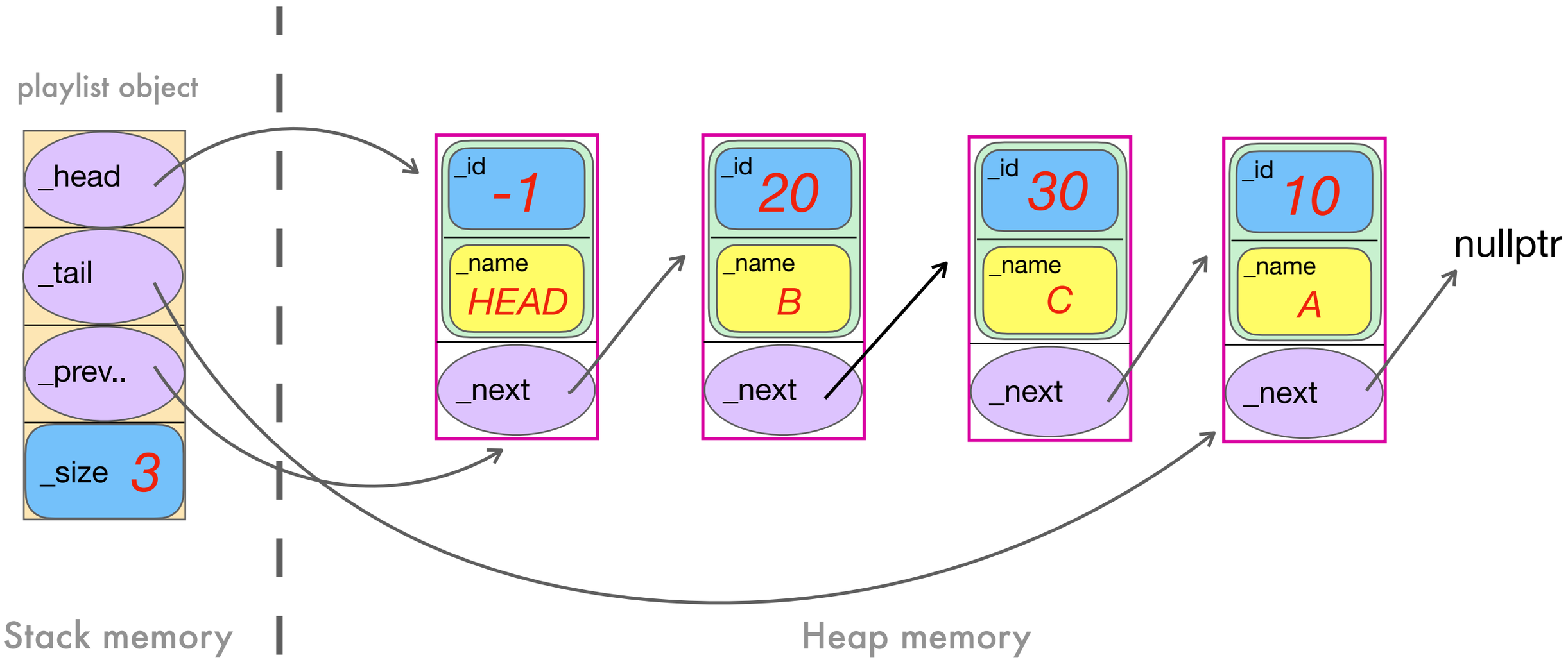


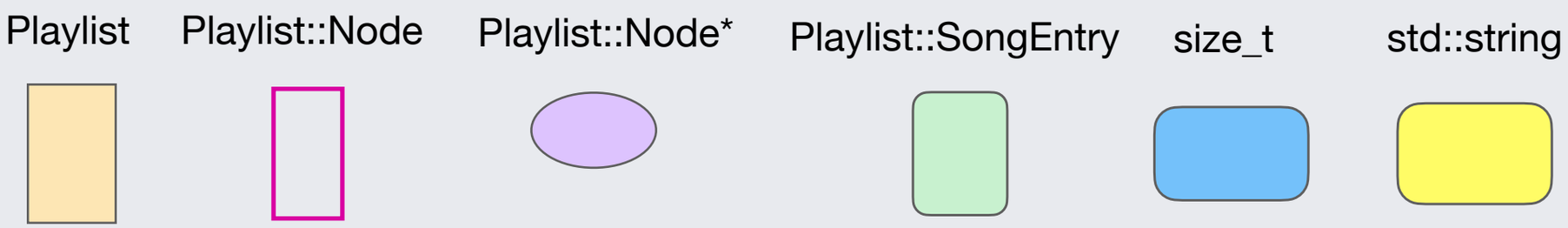
```
playlist.circular_advance_cursor();
```



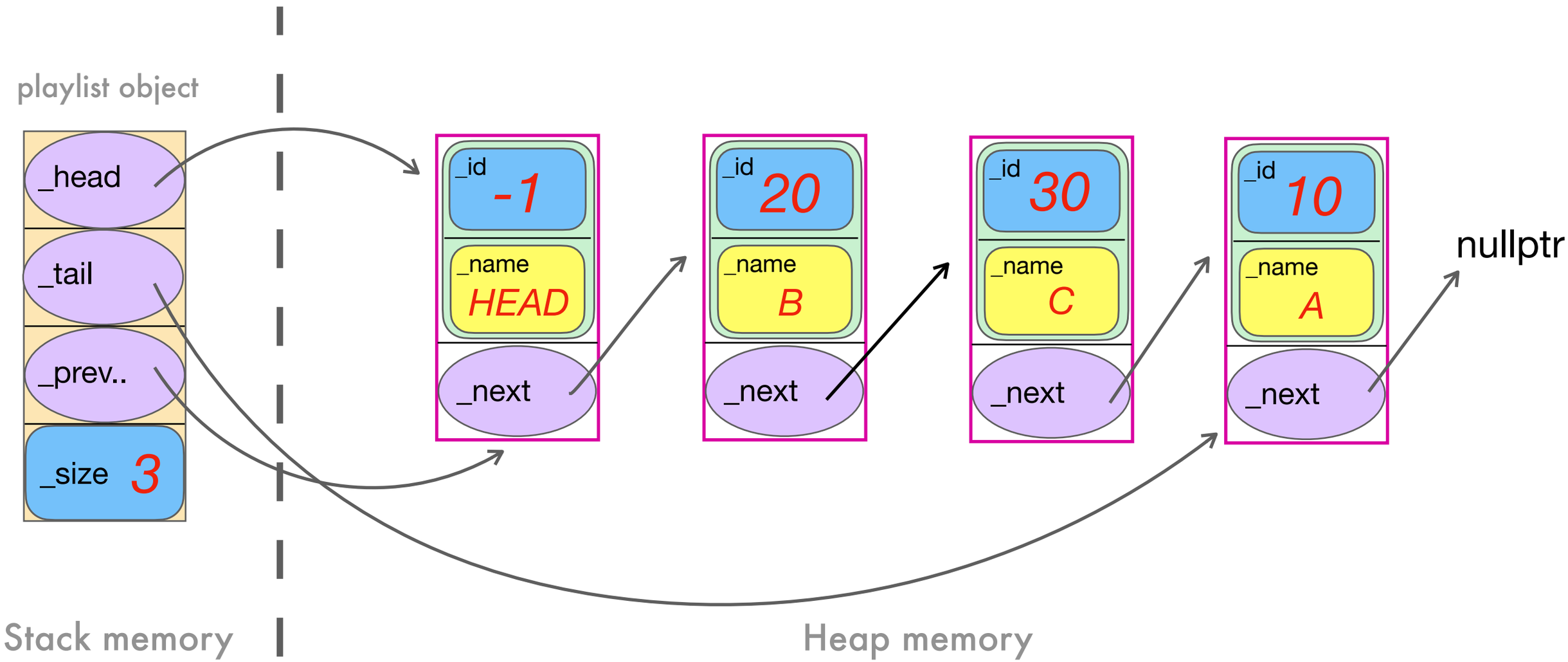


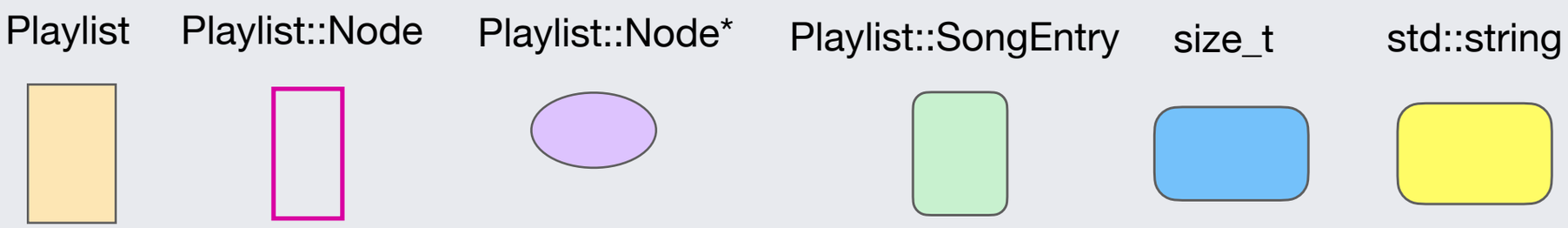
```
playlist.circular_advance_cursor();
```



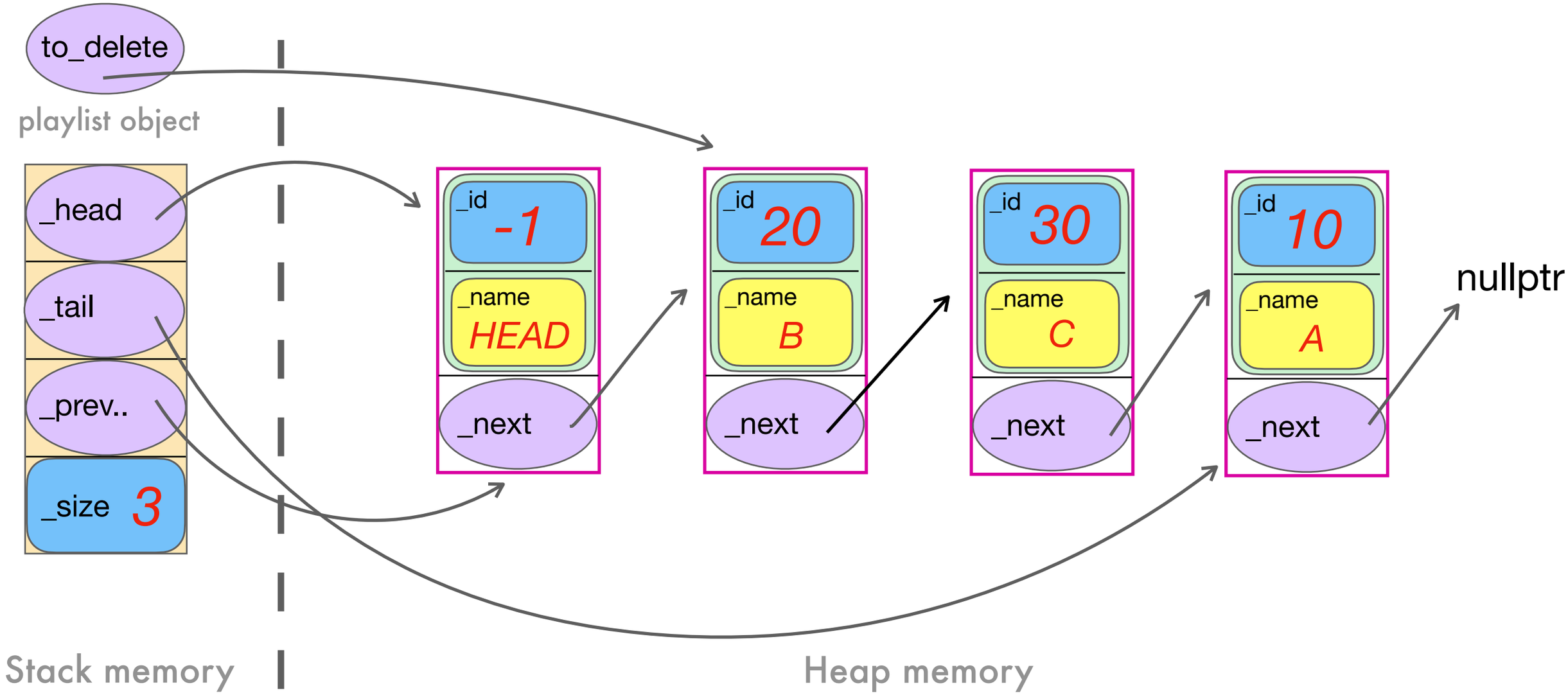


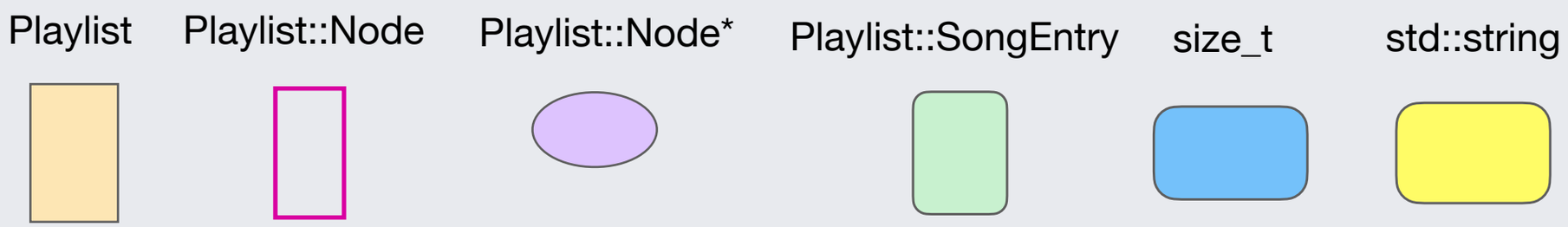
```
playlist.remove_at_cursor();
```



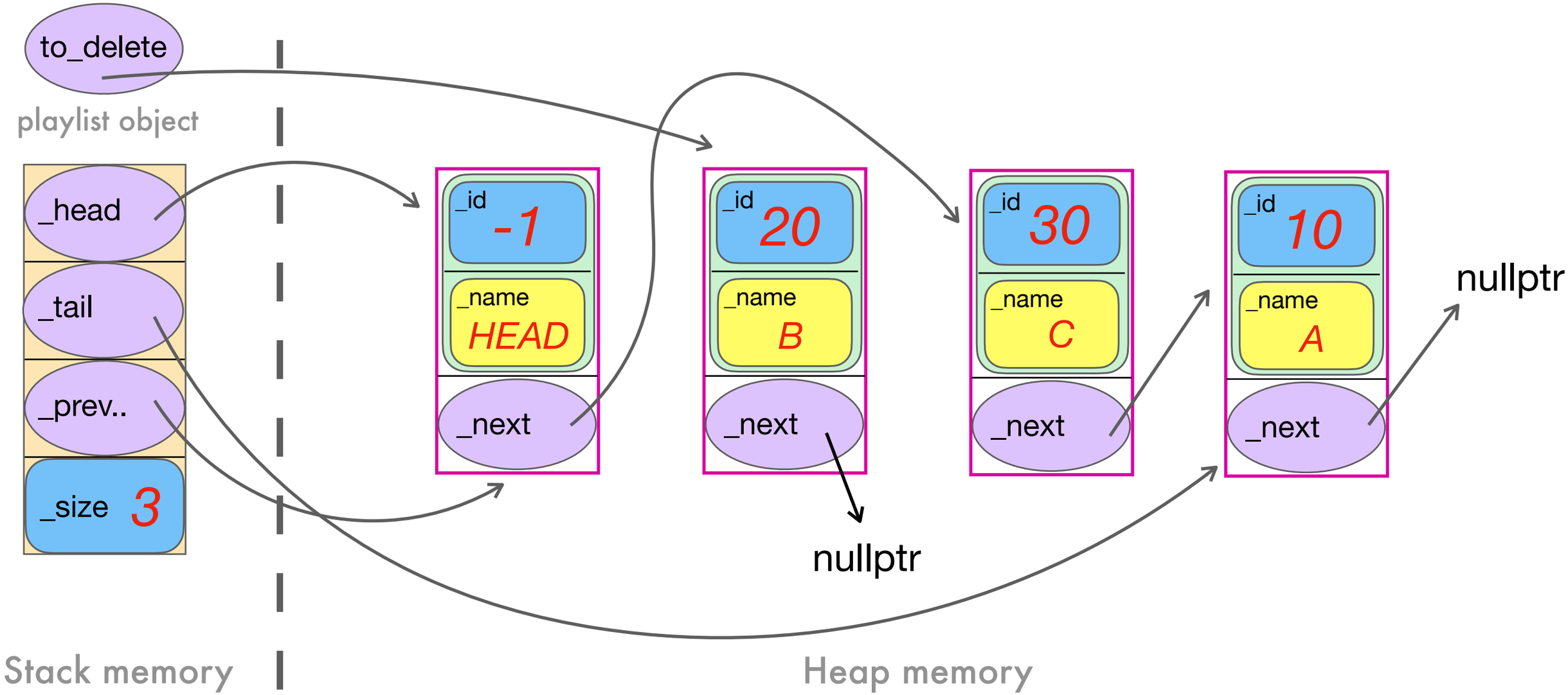


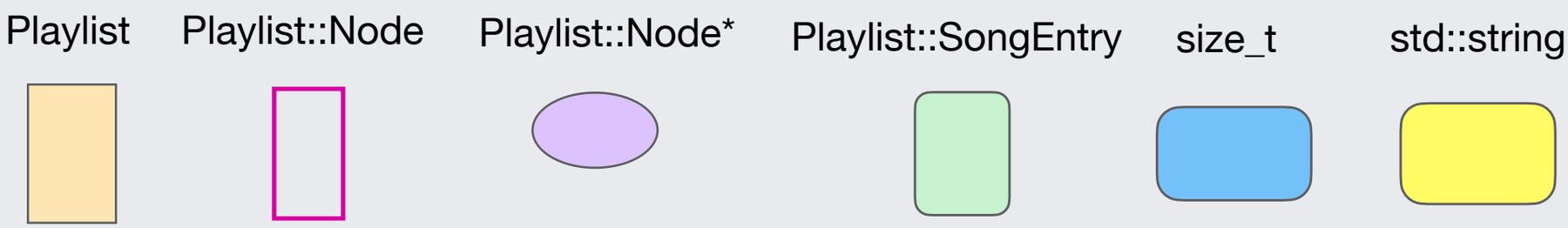
```
playlist.remove_at_cursor();
```



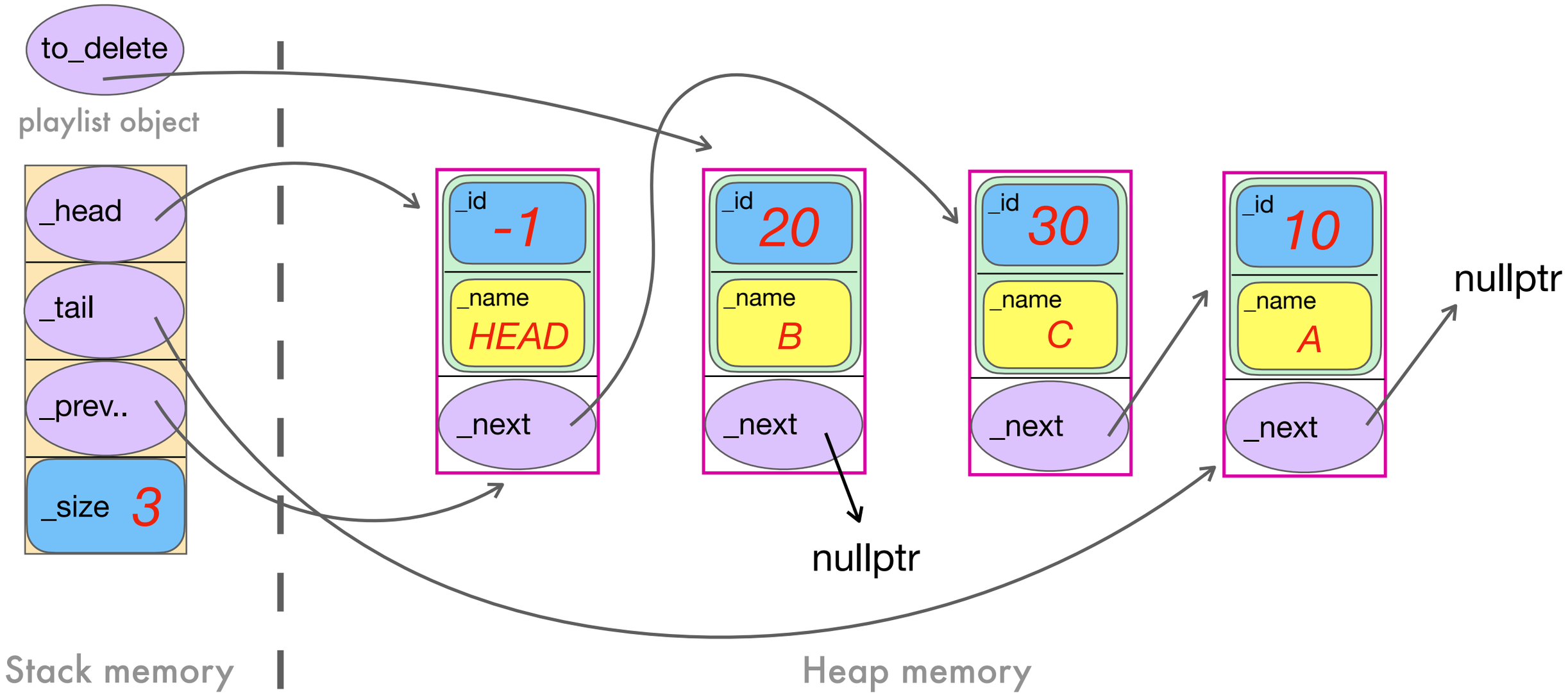


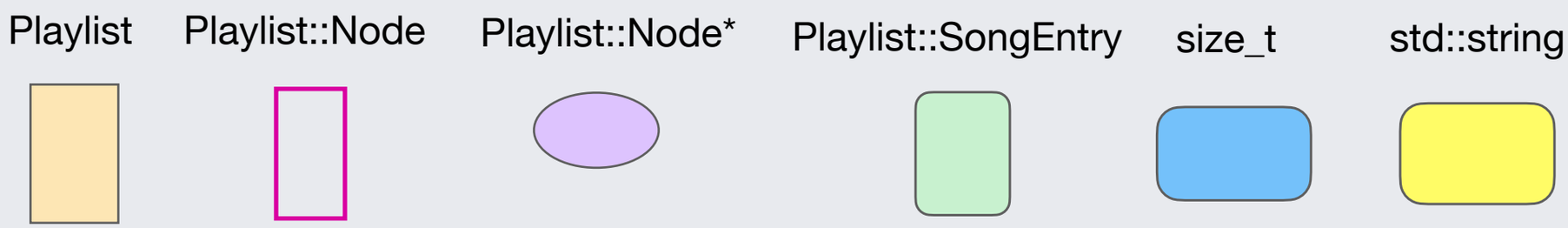
```
playlist.remove_at_cursor();
```



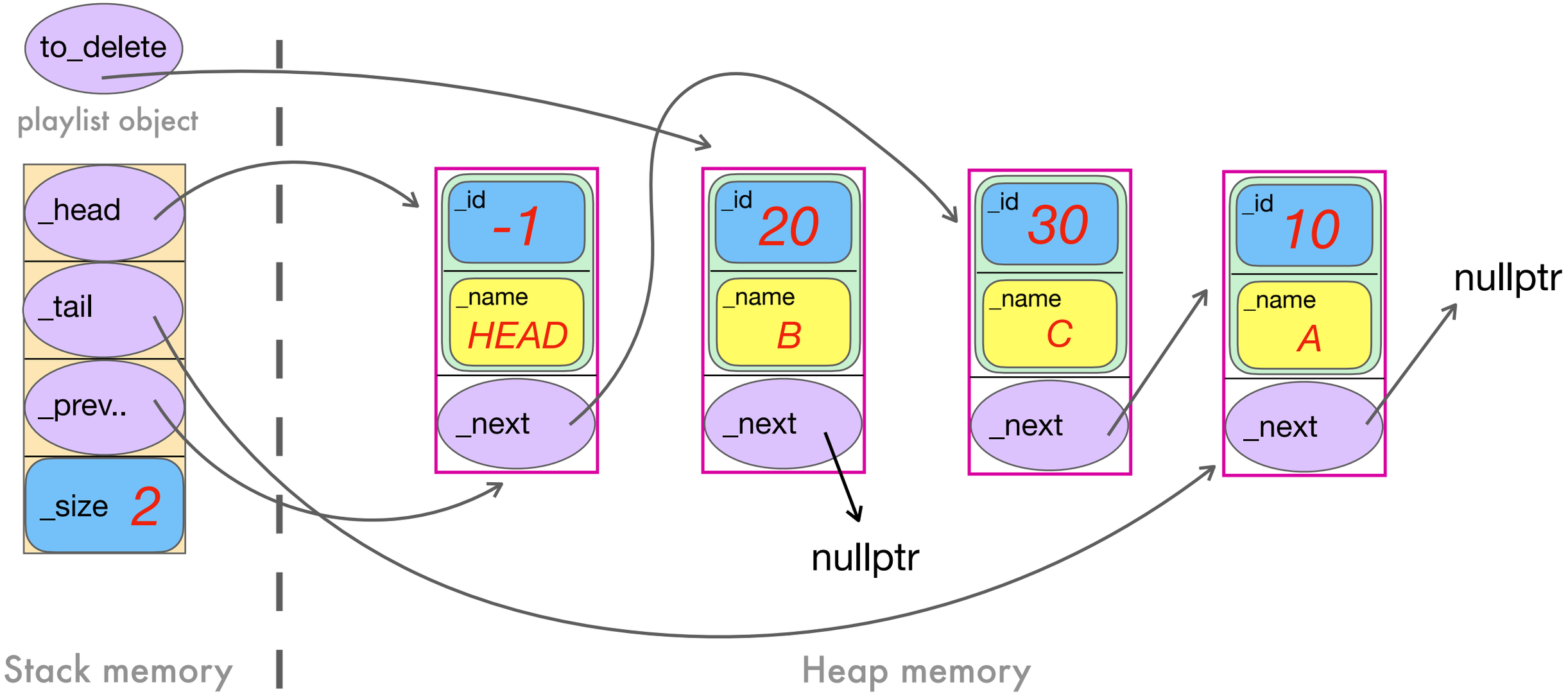


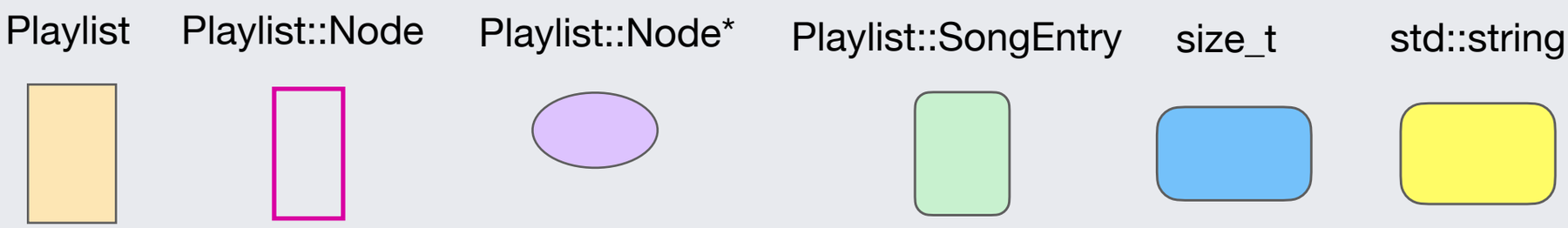
We have isolated the node to remove from the chain, and we can now safely deallocate that memory (**delete to_delete;**) and update _size.





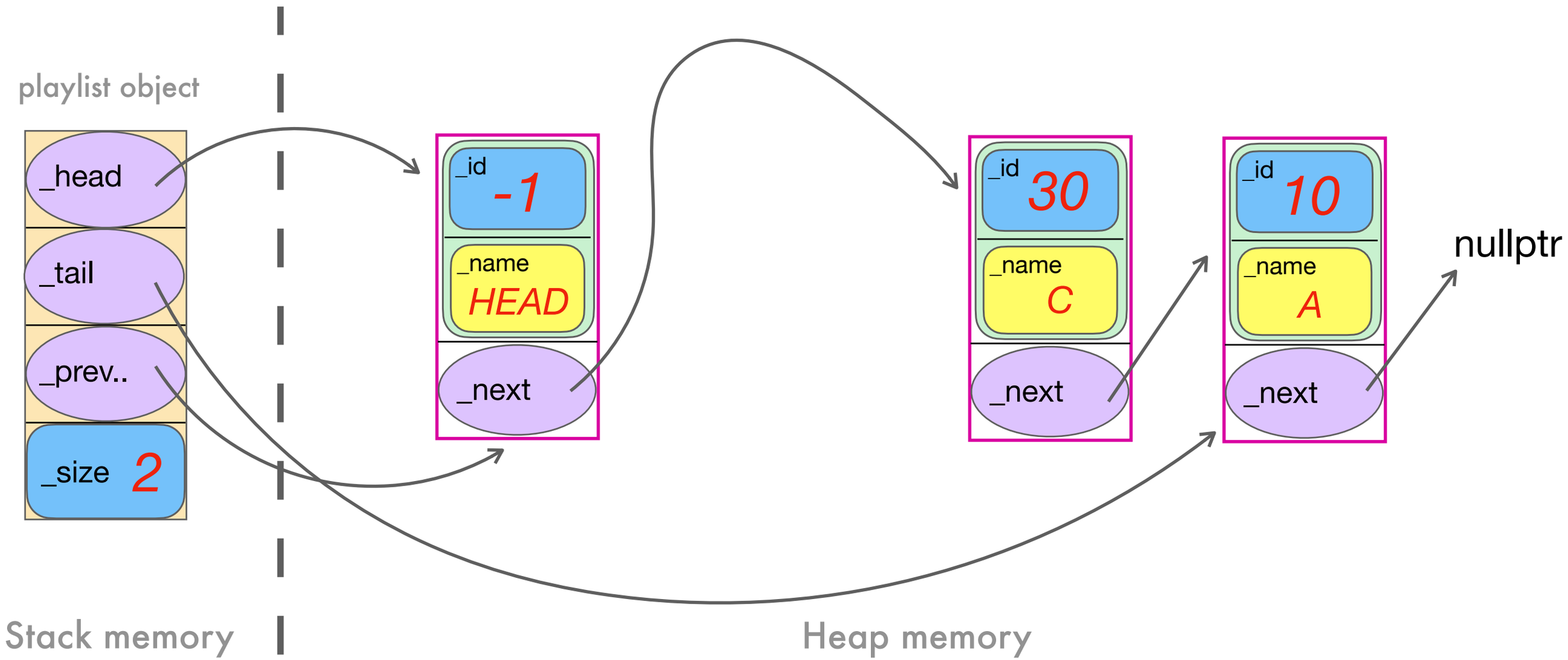
We have isolated the node to remove from the chain, and we can now safely deallocate that memory (**delete to_delete;**) and update _size.

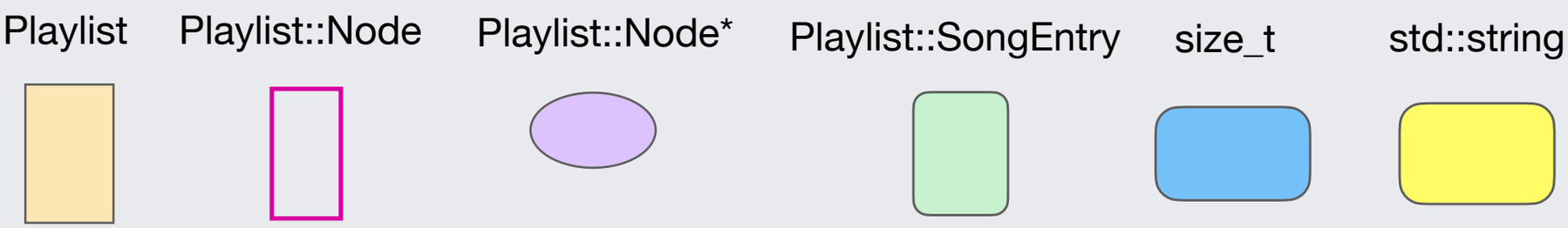




To be continued .. in your programming project 2!

Happy coding. And draw lots of pictures!





Legend